NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

STANDARDIZATION OF SOFTWARE APPLICATION DEVELOPMENT AND GOVERNANCE

by

Peter P. Labbe

March 2015

Thesis Advisor:	Raymond Madachy
Co-Advisor:	John Michael Green

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2015	Master's Thesis

4. TITLE AND SUBTITLE
STANDARDIZATION OF SOFTWARE APPLICATION DEVELOPMENT AND GOVERNANCE

5. FUNDING NUMBERS

6. AUTHOR(S) Peter P. Labbe

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

A number of Defense Department initiatives focus on how to engineer better systems that directly influence software architecture, including Open Architecture, Enterprise Architecture, and Joint Information Enterprise. Additionally, the Department of Defense (DOD) mandates moving applications to consolidated datacenters and cloud computing. When examined from an application development perspective, the DOD lacks a common approach for incorporating new technology or developing software-intensive systems that will be included in the proposed consolidated datacenters and cloud computing. This thesis will outline an architectural framework incorporating a common approach for software development based on a standard approach.

The result of this research will be a high-level guide that defines a methodology that incorporates architectural frameworks, and aligns with high-level policies and guidance to ensure more commonality and structure for software programs. This thesis shows how a common methodology incorporating commercial technology into defense systems can establish a common framework for application and technology development. This is not a simple problem to solve, but, if not addressed, DOD application development will fall further behind the commercial market. Without clear direction to the acquisition community on how to build applications, there will be a lack of alignment between strategic goals and future technology implementation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software development, SOA, software ecosystem, software architecture, software standards, commercial software development	91
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

STANDARDIZATION OF SOFTWARE APPLICATION DEVELOPMENT AND GOVERNANCE

Peter P. Labbe
Civilian, Department of the Navy
B.S., University of Phoenix, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT

From the

NAVAL POSTGRADUATE SCHOOL
March 2015

Author: Peter P. Labbe

Approved by: Raymond Madachy, Ph.D.
Thesis Advisor

John Michael Green
Co-Advisor

Clifford A. Whitcomb, Ph.D.
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A number of Defense Department initiatives focus on how to engineer better systems that directly influence software architecture, including Open Architecture, Enterprise Architecture, and Joint Information Enterprise. Additionally, the Department of Defense (DOD) mandates moving applications to consolidated datacenters and cloud computing. When examined from an application development perspective, the DOD lacks a common approach for incorporating new technology or developing software-intensive systems that will be included in the proposed consolidated datacenters and cloud computing. This thesis will outline an architectural framework incorporating a common approach for software development based on a standard approach.

The result of this research will be a high-level guide that defines a methodology that incorporates architectural frameworks, and aligns with high-level policies and guidance to ensure more commonality and structure for software programs. This thesis shows how a common methodology, incorporating commercial technology into defense systems, can establish a common framework for application and technology development. This is not a simple problem to solve, but, if not addressed, DOD application development will fall further behind the commercial market. Without clear direction to the acquisition community on how to build applications, there will be a lack of alignment between strategic goals and future technology implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AoA	Analysis of Alternatives
API	Application Programming Interface
C4I	Command, Control, Communications, Computers and Intelligence
CDR	Critical Design Review
CEO	Chief Executive Officer
CIO	Chief Information Officer
COTS	Commercial off the Shelf
CPU	Central Processing Unit
CSIAC	Cyber Security and Information Systems Information Analysis Center
DAS	Defense Acquisition System
DAU	Defense Acquisition University
DOD	Department of Defense
DODI	Department of Defense Instruction
EE	Enterprise Edition
EIA	Electronic Industry Associates
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
IA	Information Assurance
IOA	Interoperability Open Architecture
IP	Internet Protocol
IS	Information Systems
IT	Information Technology
ITS	Information Technology System
J2EE	Joint Platform, Enterprise Edition
JCIDS	Joint Capabilities Integration and Development System
JLC	Joint Logistic Commanders
JIE	Joint Information Environment
JITC	Joint Interoperability Test Center
JROC	Joint Requirements Oversight Council

LCSP	Life Cycle Sustainment Plan
MDA	Milestone Decision Authority
MILSTD	Military Standards
NMC	Naval Material Command
NATO	North Atlantic Treaty Organization
NSS	National Security System
OA	Open Architecture
OS	Operating System
OSA	Open Systems Architecture
OSD	Office of Secretary of Defense
PC	Personal Computer
PEO	Program Executive Office
PDR	Preliminary Design Review
PP	Protection Plan
PPBE	Planning, Programming, Budgeting, Execution
RAM	Random Access Memory
RFP	Request for Proposal
ROI	Return on Investment
RTI	Remote Technologies Incorporated
SEI	Software Engineering Institute
SDK	Software Development Kit
SOA	Service-Oriented Architecture
WWW	World Wide Web

EXECUTIVE SUMMARY

Department of Defense (DOD) Information Systems (IS) tend to mature at a slower rate in relation to the evolution of commercial entities (Serbu 2013). The rigor of the acquisition processes, contractual obligations and limitations, and the time lapse between contract award and the actual deliveries of systems all contribute to the lack of nimble adaptation to change within the DOD. Standards, policy processes, and the amount of time it takes to write and vet appropriate stakeholders considerably slow maturity of DOD IS.

A number of DOD initiatives have focused on how to engineer better systems that directly influence the software architecture, including Open Architecture, Enterprise Architecture, and Joint Information Enterprise (Serbu 2013). Additionally, the DOD has mandated moving applications to consolidated datacenters and cloud computing (Serbu 2014). When examined from an application development perspective, the DOD lacks a common approach for incorporating new technology or developing software-intensive systems that will be included in the proposed consolidated datacenters and cloud computing. This is not a simple problem to solve, but, if not addressed, DOD application development will fall further behind the commercial market. This thesis will outline an architectural framework incorporating a common approach for software development based on a standard approach.

Program managers need to leverage newer technologies in order to make their products relevant to the end users. Direction from senior leadership is to align with the commercial market by leveraging the latest and greatest capabilities available. Defense directives influence program managers' decisions in the execution of their product lines. These directives affect both the development and environment, and they focus either on technology or on all phases affecting the product life cycle.

By providing common guidance and standards, the DOD ensures that the objectives of the technology or software ecosystem are standard across all development environments. The DOD fosters consistency in development and interoperability when it provides a common set of objectives, a high-level view of the technology implemented,

and considerations for deployment achieves more consistency in development and interoperability. Additionally, this information will provide a level set of information that the systems engineer can leverage while designing the technology or process. Often, senior leadership dictates new or innovative technologies injected into the middle of the development cycle. The information found in the standard guidance proposed will help the program manager determine the best time and means to implement these changes.

The DOD can learn from how the commercial market defines the development environment. A common set of tools and framework allows for applications and enhancing existing capabilities to be much easier for developers. By example, creating a common set of standards and guidance provides the basis for the framework:

(1) Following common steps

(2) Providing clear guidance

(3) Libraries of application programming interfaces (APIs)

(4) Common set of services

(5) Re-useable objects

(6) Common approach

The DOD can leverage a methodology of common guidance to help ensure that program managers and systems engineers have a clear understanding of the objectives when new technology or processes are injected into the acquisition cycle. The DOD is not a commercial business; it needs to establish a set of common DOD guidelines for development.

LIST OF REFERENCES

Serbu, Jared. 2013. "DOD Brings Culture of Open Architecture to a World of Proprietary Systems," *Federal News Radio*, accessed 20 June 2014 http://www.federalnewsradio.com/394/3504114/DOD-brings-culture-of-open-architecture-to-a-world-of-proprietary-systems.

Serbu, Jared. 2014. "Navy Issues First Contract under New Datacenter Consolidation Plan," *Federal News Radio*, accessed 28 June 2014 http://www.federalnewsradio.com/412/3640559/Navy-issues-first-contract-under-new-data-center-consolidation-plan.

ACKNOWLEDGMENTS

I would like to thank my advisors, John Michael Green and Dr. Ray Madachy, for their patience and guidance during this process. I also would like to thank the many acquisition professionals and co-workers that unselfishly provided data and answered questions about the acquisition process. Finally, I would like to thank my wife, Dixie, and my stepson, Tristan, for their support and patience while I conducted research and drafted this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. BACKGROUND

A. SOFTWARE ENGINEERING

The first conference specifically relating to the software engineering discipline was in 1968 in Garmisch, Germany, under the sponsorship of the North Atlantic Treaty Organization (NATO) Science Committee (Naur and Randell 1969). The conference concentrated on basic issues and key problems in the engineering field of software engineering. One of the key topics discussed at the conference addressed whether software is different from hardware. In today's modern software world, it is hard to conceive that software engineers at the time would even consider that these two elements of a system were the same. Of even more interest is that, in 1967, the Science Committee chose the term "software engineering" to be provocative (Naur and Randell 1969). A goal was for the forthcoming 1968 conference to be able to identify present necessities, shortcomings, and trends, and that the findings could serve as a signpost to manufacturers in the computer industry (Naur and Randell 1969). Although in 1968 software engineering was at its infancy, many conference attendees agreed that software-based systems would be the future. A concern at the conference was how to ensure error-free software and documentation that provided clear direction to the developers (Naur and Randell 1969).

The initial development of the Internet started as a United States defense project called ARPANET. ARPANET started what today is the World Wide Web (WWW) or simply "the web" (Leiner 2012). As the technologies and companies supporting the Internet grew, so did expansion into the commercial market. Early software systems focused on large hardware platforms and infrastructures. Today, the commercial software market touches nearly every person and product. This is evident with Internet-enabled devices sold to the consumer market. Technologies like chat, email, video telecommunications, and Internet phone calling have changed how users purchase personal devices. In the early years, the Internet-enabled phone market proved a rich environment for experimenting with new ideas and features that rely heavily on the Internet. The subsequent maturity of these commercial capabilities is now the heart of the

commercial market. A commercial company that is first to market will typically gain the majority of the market share, often forcing competitors to develop and release solutions very quickly.

Department of Defense (DOD) Information Systems (IS) tend to mature at a slower rate relative to commercial entities (Serbu 2013). Rigor of the acquisition processes, contractual obligations and limitations, and the time lapse between contract award and the actual deliveries of systems all contribute to the slower rate-of-change within DOD. Standards, policy processes, and the amount of time it takes to write and vet through the appropriate stakeholders considerably slow the rate of maturity of DOD IS. The program manager incorporates the commercial standards once ratified. Failure to use the approved standards results in non-compliant or non-standard system designs. Commercial hardware changes too rapidly for defense program alignment. The DOD test, fix, and release process often takes longer than the release of new technology by the commercial market (Serbu 2013). DOD programs leveraging the latest commercial technology typically cannot develop quickly enough to deploy the technology before release of a new version, so the DOD issues new guidance and directives almost daily, many taking years to implement the life cycle process (Serbu 2014). An example is Information Technology (IT) Reform Act 804, which took effect in 2010 (Office of Secretary Defense 2010, 10). It was designed to completely change the way IT systems would be funded and provide more guidance to the program manager for software intensive systems. The writing of the reform act started more than five years before first release and revisions to it continue today. Despite the reform act, information systems still follow standard funding and acquisition processes used for weapons systems (DODI 2012). DOD leadership is still looking for ways to reduce the budgets with a focus on IT systems. Program managers are still building systems and applications following DOD standards and the stovepipe mindset. There needs to be a middle ground that program mangers use to make determinations of changes that make sense and that will actually save money if adopted.

The commercial application market has seen major increases in the development of common applications, specifically around platforms like smart phones and tablets.

Apple and Google (Android) found themselves having to build common platform development kits that not only allowed developers to develop products for their specific operating platform, but for applications that could easily port over to other platforms with minimal manipulation. Both platforms provide very clear guidance on what software developers need in order to create applications on specific platforms. The use and development of a software developers kit (SDK) is like a bible for development of software (Blackwell 2005). Furthermore, software development kits provide the end-to-end tools needed to develop and deploy applications in specific environments. Additionally, since there is a potential to deploy in multiple environments, the approval process provides additional details to support simultaneous deployment for each environments.

B. DOD OPEN ARCHITECTURE AND HOSTING INITIATIVES BACKGROUND

Within the DOD, there has been a major push for cost saving and alignment with the commercial market often from initial policy approval to the time a program manager can enforce the change, with new policies intending to replace the existing policy. In addition, many policies are unfunded directives that are very difficult to enforce (OSA 2014).Open System Architecture (OSA) was one of these initiatives that senior leadership required but did not provide any guidance or additional funding for implementation (OSA 2014). OSA, as defined by Defense Acquisition University (DAU), employs a modular design based on widely supported and consensuses based standards (OSA 2014). OSA is both a business and technical approach for developing a new system or modernizing a legacy system. A program manager is required to ensure the system under development follows the OSA guidelines, but no money is specifically set aside to meet the fundamental requirement of OSA. Conversely, the DOD has to retain systems for many years after procurement. Development cycles tend to be much longer than in the commercial market and the level of testing can often exceed the time to develop a system. Please apply this standard throughout your thesis. The promise of OSA as an enhancement to legacy systems or adding functionality with minimal impact to the

overall architecture is an obsolete approach with modern software development languages (Lyle 2013).

Given the requirement to use OSA, it is the responsibility of the program manager to determine the best way to implement open architecture. To allow competition without jeopardizing the existing contracting actions can be a balancing act (OSA 2014). DOD seeks incentives to collaborate with other industry partners to ensure that the government is getting the solution that meets the requirement. With the reduction in budgets and the continued change in commercial technology, DOD seeks to find ways to enhance the capabilities of the new systems built today without major changes to the underlying architecture. The use of other technologies like virtual machines and hosting can add additional flexibility to hardware architectures. Software hosted using virtual technology would allow for an abstraction layer between the physical hardware and application suite of software (operating system and applications). Development of a modular approach to the software systems would allow for ease of future upgrades for both hardware and software.

In July 2012, the DOD signed and distributed a guidance document called "Cloud Computing Strategy." This strategy intended to take advantage of cloud computing benefits that accelerate IT delivery, efficiencies, and innovation at the enterprise level (DOD 2012). The strategy included commercial best practices and capabilities for the fielding of applications. Referred to as the Joint Information Environment (JIE), this strategy aligned with the commercial cloud computing environments with the goal to save money and align IT programs. A goal of the JIE initiative was to increase interoperability across the programs and within the systems developed; however, many of the policies that came forth from this strategy stopped short of providing clear guidance at the development level, leaving the development community to rely on the commercial marketplace. Without clear understanding of the objectives associated with this strategy, program managers continue to build specific applications meet the needs of the specific user community.

C. PROBLEM STATEMENT

A number of initiatives have focused on how to engineer better systems that directly influence the system and software architecture: Open Architecture, Enterprise Architecture, and Joint Information Enterprise. Additionally, the DOD has mandated moving systems and applications to consolidated datacenters and cloud computing. When examined from a system or application development perspective, the DOD lacks a common approach for incorporating new technology or developing software intensive systems into the proposed consolidated datacenters and cloud computing. This thesis will outline an architectural framework incorporating a common approach for software development of based on a standard approach.

Thesis goals:

1. Research and explore the current acquisition processes and open architecture initiatives for the best global thinking relevant to developing applications for the future.

2. Review the current acquisition processes and templates to determine if the strategic goals align with the current processes in place.

3. Understand the impacts of incorporating commercial market software ecosystems in DOD development without clear guidance from senior leadership.

D. BENEFIT OF STUDY

The methodology of this paper focuses on research into the life cycle of DOD software policies and governance, and current commercial examples of how software policies can allow for innovation within the DOD. The result of this research will be a high-level guide that loosely defines a methodology that incorporates architectural frameworks, such as an SDK that aligns with the high-level policies and guidance to ensure more commonality and structure for software programs. The intent of any SDK document is not to be a rigid, hardened policy, but a basis for future actions regarding software development efforts. The methodology will rely on detailed research of commercial standards, DOD policies, and cybersecurity polices, narrowing focus on the guidance to development activities. Equally important, the scope of this thesis will

incorporate the research on current and past development challenges for software-intensive systems. Accordingly, this thesis will also show how a common methodology used to incorporate commercial technology into the defense systems can establish a common framework for application and technology development. Further, this research will show that, by providing more guidance up front, program managers will be more successful in achieving the goals set forth by senior leadership in the DOD.

As the DOD moves toward more complex software enterprise hosting environments, program managers, systems engineers, and system architects need to design and build new software applications using a common infrastructure. Development in a common environment also means that the owners of the infrastructure are often not the owners of the applications. Moreover, many of the resources require the application to operate successfully and provide user capabilities share resources with other applications deployed in a common environment. For this reason, it is imperative that development efforts have a common framework shared with all development efforts. Part of the framework may include cloud computing coupled with datacenter consolidation with a focus on saving money and reducing overhead across programs. Consequently, service-oriented architecture (SOA) strategies may support the alignment to cloud computing, and datacenter consolidation as a means to have a common framework for applications. A long-term cost-benefit analysis to determine if the goals are achieving the perceived goals still needs to be completed. Systems built today could very well be in use for the next 10 to 15 or more years. There are clear differences between DOD systems and commercial systems, and understanding these differences will help leadership make better decisions about which commercial strategies and technologies fit best within DOD acquisition processes.

E. SCOPE, LIMITATIONS AND ASSUMPTIONS

The focus of this thesis is on technology insertion during application development, leveraging commercial best practices. It assumes the level of direction followed with regard to infrastructure and architecture is common for all applications in a single environment. The thesis will touch on the importance of strong architecture

documentation and validation. This is not a simple problem to solve, but, if not addressed, DOD application development will fall further behind the commercial market. Without clear direction to the acquisition community on how to build applications, program managers will continue to build products as standalone applications. A program manager's focus is on cost, schedule, and the performance of each individual program. This paradigm does not lend itself to sharing data or reusing other products to meet their objectives. Typically, commercial companies, as prime contractors, focus on the business aspect of development, profit, continued work, and reputation, not necessarily on getting products to market rapidly. The technical side of this problem is much easier to solve.

Many of the technical strategies discussed in the research of this paper are a thesis topic in their own right; implementation cost, return on investment (ROI) analysis, and justifications are a few topics that lend themselves to additional research. For the purpose of this thesis, the total life cycle cost and ROI are not considered. Rather, this paper will focus on the impact to change once the acquisition community has accepted an agreed requirement and transition to the acquisition process. This paper focuses on the impact to changes made once system development has started and changes to the process either for technology reasoning or by senior leadership directives. Life cycle cost and ROI of implementing change into the development cycle of the acquisition process can be even more difficult to determine once the Analysis of Alternatives (AoA) process is completed.

F. SOFTWARE ARCHITECTURES AND FRAMEWORKS

Software has changed over the years. The complexity of software-defined systems continues to increase. Size and complexity of the systems also continue to increase, and the design problems go beyond algorithms and data structures. Protocols for communications; synchronization and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among alternatives are all critical aspects of the software architecture (Garlan and Shaw 1994). Moreover, software is no longer a simple engineering problem that a few engineers in a back room work on. The development of software starts with a clear understanding of the architecture. Architecture is as much

about documenting the detailed aspects of how the software expects to meet the requirements as it is about defining system-coding methodology. There are many styles of software architectures, and they continue to expand as different languages and data requirements change the software environment.

Software engineering continues to evolve like many technical fields today. The phrase "a software ecosystem" may be a new term to some, but in today's software-intensive environments, software starts to become an ecosystem of its own (Bosch 2009). A software ecosystem consists of the set of software solutions that enable, support, and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions (Bosch 2009). With the evolution of the computing industry and a transition from mainframe computers to desktops (and now the mobile market), we see software ecosystems support this growth. This thesis will touch on how software ecosystems' growth influences the defense market at all levels.

Figure 1 depicts the Java Enterprise Edition (JEE) architecture. It is an example of a reusable architecture. The design is simple, easy, and logically organized into reusable components. The use of underlying services in the form of containers helps to ensure re-usability (Oracle 2010).

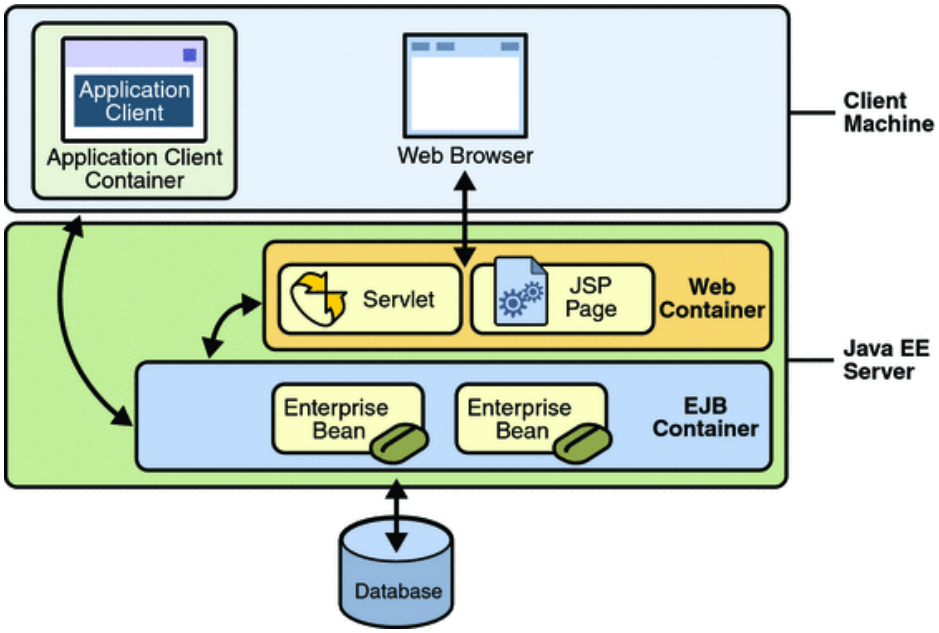

Figure 1. Java EE Server and Containers (from Oracle 2010)

Defining the architecture is not the only aspect that needs consideration before any development can take place. A software framework commonly defined as "a platform for developing applications. It provides the foundation on which software developers can build programs for specific platforms" (Techterms 2013). When properly defined, the framework helps the programmers to ensure that the software works with the hardware platform and the various aspects of the software. Another way to think about the SDK framework is to consider the base Operating System (OS), which typically comes with all the tools needed to develop in that platform. Java Platform, Enterprise Edition (J2EE), specifically the "JavaBeans" Framework, used to illustrate the framework construct. JavaBeans are re-usable software components for the Java language (Janssen, 2014a). The JavaBeans concept allows the encapsulation of multiple objects into a single Bean. Beans register to receive or send objects to other applications or other parts of the system. A program can re-use the Beans or objects in multiple Beans depending on how the application works.

Staying with the Java EE example, many applications have multiple tiers in which the logic is distributed. Figure 2 provides an example of the Java EE multitier approach.

Couple this with the use of the containers in Figure 1; the developer now has a multitier re-usable architecture (Oracle 2010).

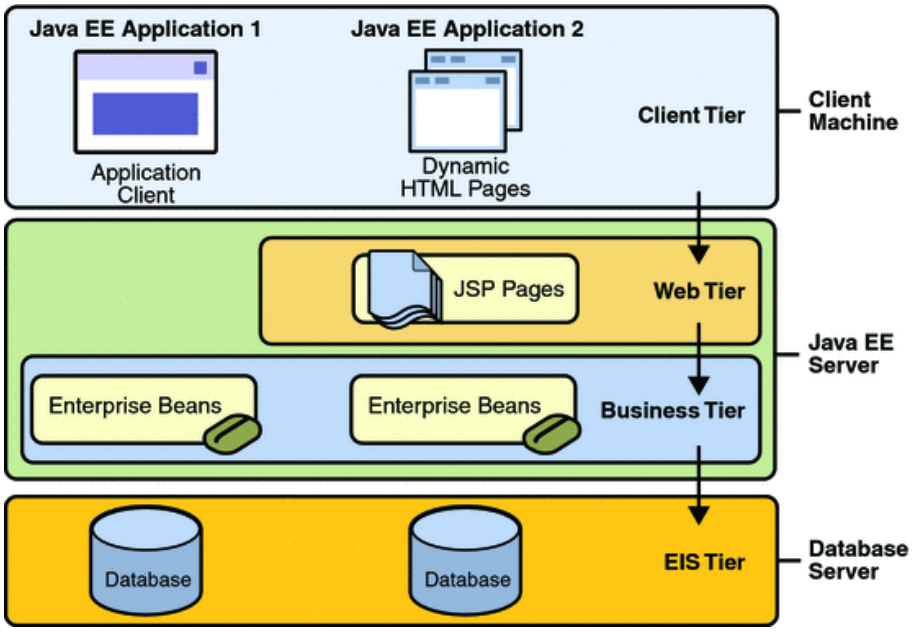

Figure 2. Multitier Applications (from Oracle 2010)

Java multitier applications, generally considered, three-tier applications because they distribute over three locations: client machines, JAVA EE servers, and databases or legacy machines at the back end (Oracle 2010). As with any software architecture, there are a number of items considered for any development. Java EE is a re-usable container and modular approach that provides clear guidance for usage. Though not called an SDK, it contains many of the same attributes. Commercial guidance and lessons learned are often the best tools that a developer can rely on. Incorporating the commercial standards from design through development allows the architects and engineers to remain on the same page. Java EE web guidance is a good example of the types of information needed during the development phase.

Architects need to define tools and constraints before systems engineers or software programmer's start building the system. Two key aspects are the architectural design and the proposed framework. As strategic plans change how software systems will operate in the future, there needs to be more consideration at the software development

level for how these two aspects, architecture and framework, will change what program managers can develop or how they need to change the existing systems that they are developing. A good example is a legacy system originally designed to operate on a standalone server environment with specific software, such as a Microsoft OS (not the .NET environment) deployed to a datacenter. If the original system not designed for datacenter hardware, interfaces may not access the network environment making the system unusable. Consequently, the application may simply not work, require many changes, or require extensive re-writing of the application logic. This is typically outside of the program manager's control as the original plans or sustainment funds are not sufficient to accomplish the changes required.

G. THE "ILITIES" OF SOFTWARE DEVELOPMENT

Unlike much of the commercial market, DOD software development requires sustainment for many years (Serbu 2013). Commercial companies can build new updated systems or applications and release them to the commercial market very rapidly, as seen by the number of releases in a given year. Users wanting to upgrade simply buy the new products. At some point in time, typically after five years, commercial companies stop supporting older versions of their systems or applications. DOD systems do not have the luxury of replacing systems at the same pace as commercial companies. DOD has to maintain the supportability, trainability, interoperability, compatibility, usability, and other aspects of the systems and applications for many years. The cost of building a system or application is only part of the cost of the system or application life cycle. Program managers need to build products that can managed for many years under ideal circumstance, and upgraded as technology changes. In contrast, commercial development efforts provide support for multiple applications running within the same framework. The commercial market does not focus on the "ilities" of the system per se. In the DOD world, interoperability is often mandated. Many Command, Control, Communications, Computers, and Intelligence (C4I) or weapons systems applications must share data, exchange messages, access, and utilize information from common databases. DOD often defines guidance only at the highest levels of the development environment; however, by changing the role of the environment to a common framework and methodology, other

"ilities" like compatibility, usability, and commonality start to take on a new meaning. There may be an argument that these are all part of interoperability, but from the development perspective, the issue of resources and databases is only a small concern. Life cycle support, training, and maintenance boundaries are just as important. As program mangers try to build a life cycle budget, they need to understand what aspects of the common infrastructure they will be responsible for and how this will be coordinated across the life cycle of the entire system of systems. While DOD may be able to incorporate some lessons learned from the commercial market, much of this will be new and negotiated as the common environments are developed.

The DOD has defined a high-level enterprise architecture with which all services and agencies must comply; however, they still need to define the standards at the applications level. Guidance needs to be provided that aligns the strategic plans to the development community. Interoperability considered in the beginning of the engineering life cycle through use of a common framework removes guesswork from the design. Compatibility, continuity, usability and life cycle maintenance are afterthoughts to the capabilities sought. Unless there are clear goals of interoperability and clear governance on how systems work, individual systems will be built as closed or standalone systems. An example is an Enterprise Resource Planning (ERP) system that each agency or service is currently developing; without a common set of guidance and goals, there will be no incentive to ensure interoperability and common database structures across the services or agencies.

H. CHAPTER SUMMARY

This chapter discussed a number of key challenges facing program managers at the acquisition level and within the development cycle. Program managers often do not have years to change the direction of their development efforts. Many of the current frameworks and software ecosystems have been in place for many years. Program managers typically incorporate what is available at the time of execution for a development effort. The reality is that cost, schedule, and performance drive the direction the program manager will lead his team. As important as the requirements are for understanding the user needs, derived requirements are just important to capture for the environment and associated needs of the

system. Derived requirements found in the "ilities" of a system drive the cost for the life cycle of the program. Adding commercial technologies into a current development effort can cause extensive cost and time delays to the program. In fact, finding a balance and ensuring the sustainability of the system is the primary job of a program manager, even if means not following the latest senior guidance.

Chapter II addresses some of the mandates and expand on how the guidance and its impacts on the acquisition process. Chapter II discusses the requirement process and the potential impact to the development effort when changes occur. Chapter III focuses on the acquisition aspect of how Chapter II's efforts affect the development of applications. DOD has a clearly defined process from the user request for capabilities to the development of requirement. Program managers have to consider all aspects of the program including requirements, security, and life cycle support. Unfunded mandates and changes once a program is in the sustainment can be very costly and typically not budgeted. Chapter IV provides a view from the commercial development frameworks. Commercial development typically follows clear guidance and is often easier to use than the DOD processes, and Chapter IV explains why and how this could actually help the DOD development community. Finally, Chapter V concludes the paper. Chapter V presents the argument with analysis demonstrating that the finding of this research supports the recommendations. Chapter V also presents topics for additional study in the area of unfunded mandates and ways to improve the acquisition process.

THIS PAGE INTENTIONALLY LEFT BLANK

II. INFRASTRUCTURE AND DEVELOPMENT CHALLENGES

A. INTRODUCTION

Chapter II discusses directives that the defense market needs to follow when developing a new software intensive system or adding to an existing software system. Defense directives can be technology focused, such as the Joint Information Environment (JIE), or implementation specific, such as cloud computing. Directives can also be implementation-specific like Open Architecture (OA) or interoperability. This chapter describes the importance of these directives as well as some of their shortfalls. Defense guidance typically addresses the milestones leading to development and post-development phases. Many directives focus on commercial market as the guidance; however, not all commercial development and implementations will be applicable in defense environment. Where relevant, this chapter will compare and contrast the DOD market and the commercial market.

In the commercial market, changing technology is a common occurrence. The commercial market based on making money and being first to market. Having a strategy that allows companies to capitalize on the latest and greatest technology results in profits and market share. For DOD, being first to market is more about providing a critical capability that the users need to stay ahead of their adversaries. Sharing of data and use of common tools provides synchronization across the services. Program managers need to leverage newer technologies in order to make their products relevant to the end users. Direction from senior leadership is to align with the commercial market by leveraging the latest and greatest capabilities available. Datacenter hosting, cloud computing, and open architecture are a few of the highly touted commercial capabilities that senior DOD leaders are directing. A major challenge for DOD entities is that many directives come without strategies or implementation plans, resulting in program managers defining their unique solutions in a vacuum. This chapter presents a case study of a commercial company's transition to cloud computing. Similar methodologies could provide DOD program managers the needed guidance for implementing commercial technologies in

DOD infrastructures. The case study helps to build on the process flow described in Chapter I.

B. DEPARTMENT OF DEFENSE GUIDANCE

The alignment of strategic goals to organizational goals is a major contributing factor to the failure of many programs (Brownsword et al. 2013). Software development is subject to many different types of requirements. The program manager needs to control all aspects during all phases of the program. For example, requirements creep and changing requirements are major contributors to the failure of many software development efforts. Software intensive systems are especially susceptible to requirements volatility. Software requirements decomposed into specific functionality drive the capabilities of the software systems. Having a clear set of requirements based on the desired capabilities of the system is a critical design factor many program managers need to understand early in the development process. Furthermore, Brownsword et al. (2013) point out that during the major decision process, there needs to be a close relationship between the software architecture and the acquisition strategy. In fact, they conclude, not having this relationship often leads to misalignment resulting in program restarts, cancellations, or failures.

Brownsword et al also describe a framework for a common approach to software development. This approach, summarized in Figure 3, has been in place for many years (Defense Acquisition System [DAS] 2015). Program managers currently follow the standard acquisition process. Therefore, the real question is why many programs have issues and fail. Part of the challenge is not how the process works, but to what level is standardization understood. All program managers and acquisition professional have to have a certain level of training. They all have to understand the role and responsibilities for developing hardware or software systems.

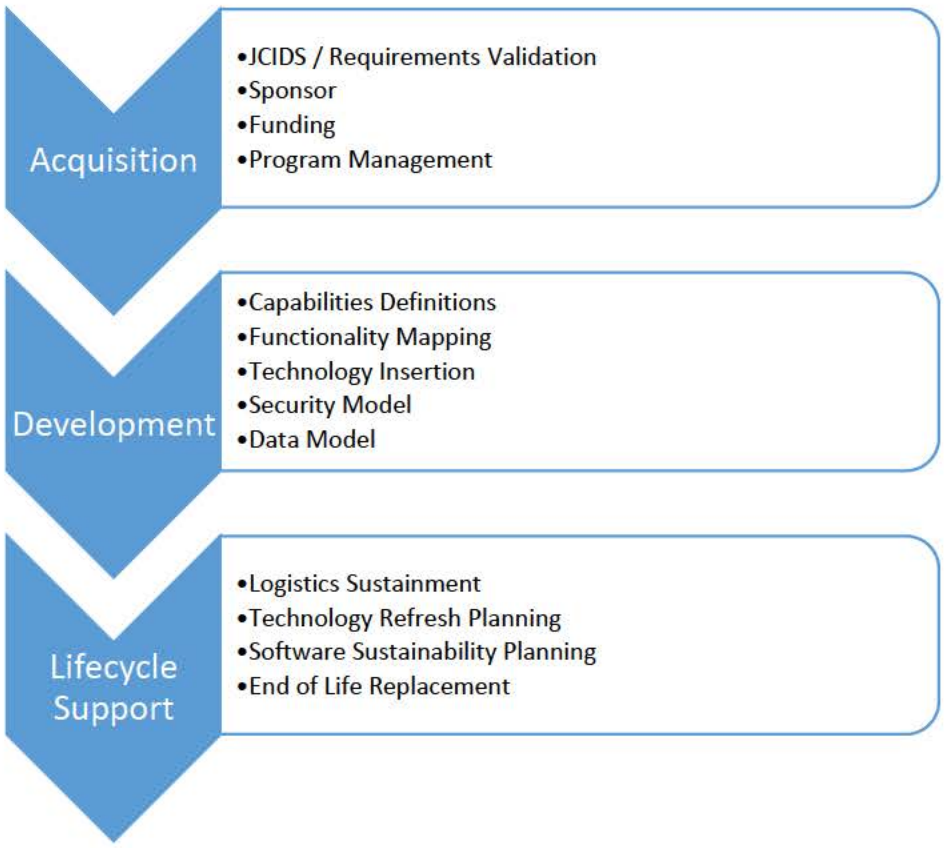

Figure 3. Simplified Acquisition Process (after DAS 2105)

In April 2013, all agencies within DOD who built, deployed, or used information systems were required participate in the Joint Information Environment (JIE) as part of the enterprise architecture (Takai 2013). Moreover, in recent years, senior leadership issued multiple directives that changed the basic framework of software development. Many development efforts have multiple year contracts and the DOD acquisition system is not aligned to rapid changes like those found in the commercial market. Therefore, program managers typically have two choices: continue to build based on original requirements and framework, or modify the framework and delay the development process in order to meet the latest directives.

Figure 4 is from the "*Guidance for Implementing the Joint Information Environment*" and shows the different phases of how JIE implementation happens. This Department of Defense guidance document establishes the framework for the JIE

framework. In the document, authors discuss the fact the JIE is not an acquisition program; however, the focus of JIE is to "consolidate, standardize, and optimize" programs and to ensure that they comply with the applicable directives (Takai 2013). JIE is to influence and affect outcomes of the acquisition process for programs under development. The guidance document points out there are multiple ways to implement the JIE strategy. There is no clear direction to the acquisition commands or to the program managers; it only suggests that they monitor during the milestone reviews to evaluate if programs are in alignment. Without direction on how the development should align, many systems will continue down the path of least resistance and continue building stovepipe systems that reside in and share a common infrastructure. This allows other applications and systems the ability to see data or services within the enterprise. There is limited guidance to what this means or how to implement these kinds of changes with a given system. In the last paragraph of the JIE Guidance document, there is a mention of funding to allow existing applications a migration path to datacenters. Missing is how to repair or upgrade those applications not initially designed to work in that environment or configuration.

Figure 4. JIE Implementation Phased Approach (from Takai 2013)

The JIE roadmap is the key aspect of Figure 4; it provides insight into the planning and implementation of the initiative. Program managers continue to align program development based on the common JIE framework. The JIE framework is one of a limited number of senior-level strategies that provides some level of guidance toward implementation. The JIE strategy is based on standardized architecture and common services. The initial introduction of the strategy garnered a lot of support. Since the implementation plan will take many years to execute, DOD will not realize success for many years.

C. DATACENTER AND CLOUD COMPUTING

Datacenters and cloud computing incorporate many of the same technologies and are often closely related; however, they have different functionality. Though most cloud computing centers reside in datacenter-like infrastructures, the ability to support stateless (does not keep track of configuration settings, transaction data, or other information) computing is not the role of a datacenter. In the article "Clouds are not Datacenters," Bias describes the differences between datacenters and cloud computing (Bias 2008). Datacenters are typically a single building specifically designed to house computer systems or telecommunications systems. Datacenters generally include redundant power and back-up power. Cloud computing is an abstraction of a facility. Typically, multiple datacenters are required to ensure cloud resources are always available. Some similarities are that both datacenters and cloud computing offer specific resources, storage, memory, and CPU power. Cloud computing users do not care where these resources come from. A datacenter is a specific location and limited in the offerings by what is located in that building. Bias explains that distributed computing is a key aspect of cloud computing which defines the relationship to application development. Development of applications destined for cloud computing requires detailed architectural forethought and detailed design work prior to fielding. Legacy standalone applications are not necessarily good candidates for implementation in a cloud environment. Application architecture designs are required before development can start. Cloud applications are specific to the environment in which they reside. Migration of an existing application to a cloud

environment typically requires added cost and time to ensure operability. The JIE strategy is an example of DOD's plan to migrate to a cloud based architecture. This strategy is very complex. However, many commercial companies have made this same transition. The following case study of Fujitsu is an example of this transition. Fujitsu built a three-phased plan, which provided very clear guidance and execution strategy. The result was a methodical plan, which resulted in cost savings and new business markets.

Cloud Computing Case Study

Fujitsu has two different business models in which they use their cloud computing technologies: offerings to the public sector, and internal software development environments. This case study will show how a common approach helped to establish clear objectives for the company. This case study also shows that, with the right methodology, the migration to cloud computing can be profitable and can improve internal business processes. Fujitsu had three issues: increasing server operations cost, server over/under utilization, and increasing labor hours for constructing development environments.

1. Increasing server operation cost: The development centers all buy, manage, and operate their own hardware as standalone platforms. Estimations that maintaining the servers required 1.5 person-days per month per server. As the number of servers increased, the labor hours to maintain the servers increased (Arimura and Ito 2011, 325).

2. Server over/under-utilization: Testing conducted during peak utilization periods supported the addition of more servers to meet the needs of the development teams at each location. This left under-utilization periods and inefficient usage of the resources during idle time.

3. Increasing labor hours for constructing development environments: Middleware developers utilized independent hardware suites to support each development environment. As the number of middleware products increased, so did the number of single-focused hardware platforms to support testing. Each platform required additional resources to manage and support the hardware and software environments. With the addition of 64-bit CPUs and virtualization software, the number of platforms for testing products increased dramatically. This last problem showed an eightfold increase in the number of test platforms required between 2005 and 2009, from 10 to 84. During this same time, the labor hours needed to build a product, construct a test environment, and test the product showed

a 25% increase from 2005 to 2008 and then another 30% increase from 2008 to 2009. Because of the increase in labor hours, the cost of doing business, and the changing technologies, Fujitsu made the decision to use their own environment to support their own development (Arimura and Ito 2011, 326).

4. For three years starting in 2008, the Numazu Cloud Center undertook a conversion to a cloud-based software development environment. Fujitsu developed a three-phased methodology based on lessons learned from previous efforts. The three phases, consolidation and virtualization, standardization, and systemization supported software development efforts as well as the commercial hosting offerings as described. Figure 5 shows the three phases and the list of objectives for each. Fujitsu used a detailed approach breaking down objectives for each phase before moving to the next phase. The case study explains how each step leads to the next but only after the objectives achieved.

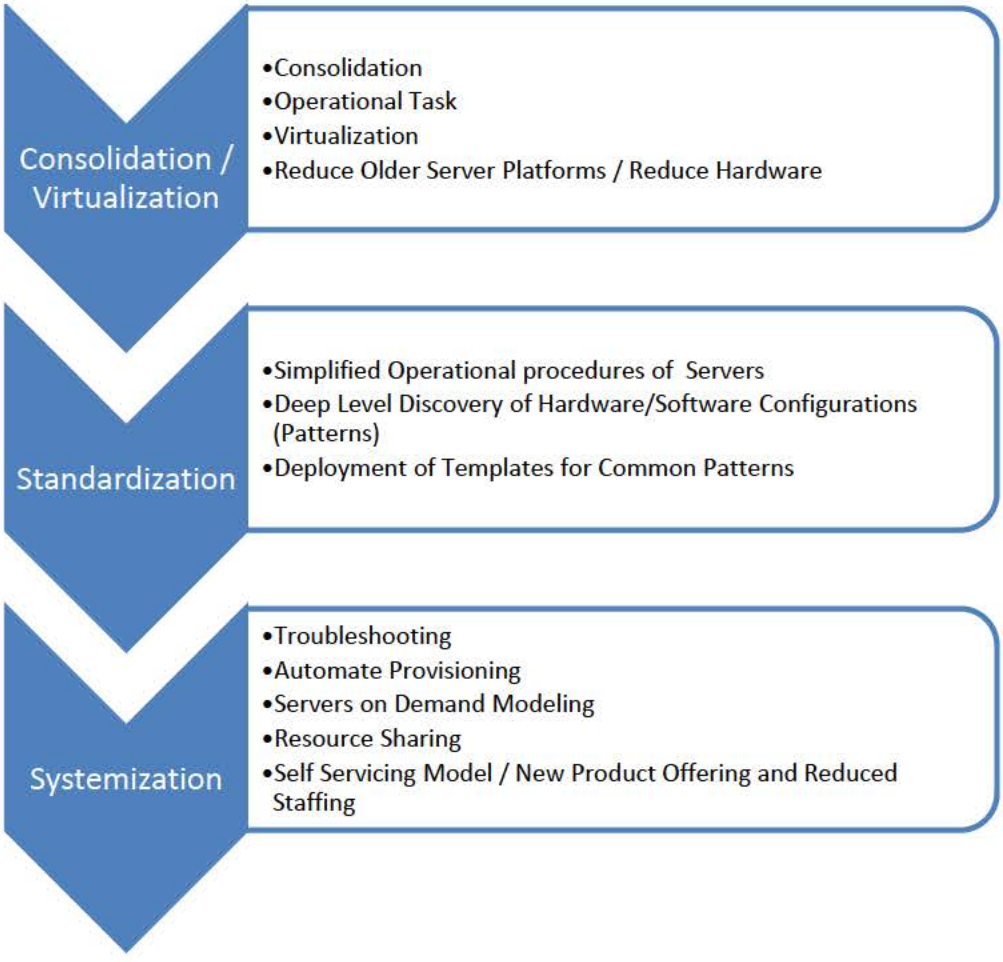

Figure 5. Cloud Computing Methodology

Consolidation and virtualization: during 2008, the company started the consolidation of the servers, initially targeting about 1800 servers, later realizing only about 100 really need to be consolidated. With the use of virtual servers, the middleware developers could then rent actual machines or virtual machines. Because of the nature of the development, most developers needed actual machines. Between 2008 and 2010, the number of virtual machines increased from 900 to more than 2300. This consolidation and use of virtual machines was not without challenges. The company had to add additional hardware and special software to handle the specific needs of the developers. The result has been a more efficient development environment (Arimura and Ito 2011, 326).

Standardization: Based on analysis of a 2008 study of their virtualization environments, Arimura and Ito (2011) discovered that the company had 348 patterns, each representing different combinations of CPU number, memory size, disk capacity, and OS type. According to the same report (329), approximately 51% of the patterns found to be very similar.

Systemization: The software development partners continued to increase as other companies used the Fujitsu cloud services as their development environments. Arimura and Ito (2011) report shows the demand on resources continued to increase, placing a burden on manual processes to provision the resources needed to support the dispersed development teams. In order to meet these needs, Fujitsu developed a number of products to automate the processes. This included a service catalog of products; automated deployment environment; automated operations; dynamic resource management; and automated operations in a cloud environment.

Fujitsu offered these products to the commercial market as part of their new business line of cloud computing services. They also used the new cloud environment to re-focus their internal development teams. The lessons learned from the internal development processes helped Fujitsu to continue to improve their external commercial offerings (Arimura and Ito 2011, 329).

Fujitsu made an initial invested of $14 million in hardware and software over a three-year period. As of 2011, Fujitsu projected a continued cost reduction of $9 million annually for infrastructure and $2.5 million annually from terminating leases and consolidating services. With the deployment of the new software management tools, Fujitsu balanced utilization of servers and realize a significant reduction in labor hours to maintain and manage their cloud environments. By reducing hardware and using newer technology, Fujitsu has seen a reduction in power consumption, which added to the total life cycle savings for the company.

In summary, Fujitsu's initial approach required modifications and development of clear goals. By creating their three-phase approach, Fujitsu established a detailed methodology to transition from standalone environments to cloud computing. Fujitsu took advantage of lessons learned from other companies that had made similar transitions. Fujitsu also expanded the lessons learned to include how they defined their internal development environment and commercial offerings.

This case study provides clear examples of how defining a clear methodical approach to cloud computing can help a company to achieve cost savings and improve business opportunities. For DOD entities, the lessons learned can help define clear methodologies that organizations can leverage when moving major applications or reducing facilities with a cloud computing initiative. Fujitsu showed that it is not only possible to reduce cost and hardware, but to expand internal development capabilities critical to business operations as well Figure 6 provides an example of a model derived from figure 5, that can be used in DOD to help program managers better manage the transition to cloud computing.

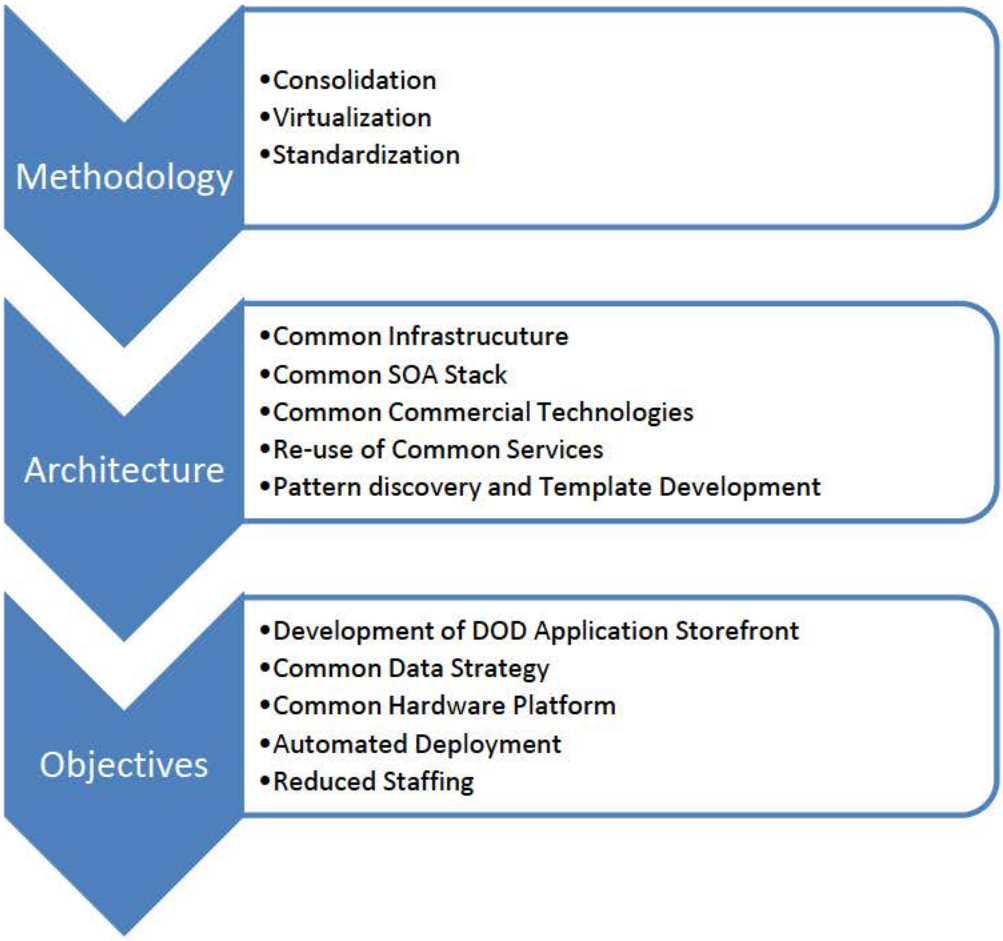

Figure 6. High Level Cloud Computing Deployment Model

Based on this simple approach, the senior levels of DOD that direct the program managers and acquisition commands to move to cloud computing could do so with a common set of standards and objectives. Technology is only one aspect of using commercial technology. DOD needs to consider the methodology that program managers will use to leverage these technologies and provide clear guidance on the objectives. The Fujitsu case study showed that by defining a three-year plan, laying out clear tasks for each year, and then empowering the program managers with execution of the plan, they were able to achieve and even exceed the initial goals. These are lessons learned that DOD should take advantage of as they move to cloud computing.

In 2010, the U.S. Chief Information Officer issued a paper titled "*25 Point Implementation Plan to Reform Federal Information Technology Management*" (Kundra

2010). Many of the points in this thesis are contained in the 25-point plan. Like many management plans, focusing on how to align the money with the execution is a critical aspect of the management strategy for this effort. In the paper, the article tried to align the contract language to the Request for Proposal (RFP) process. Parts of the paper were very inspiring, and it appeared that much of the argument of the paper aligned to the execution of the management plan; however, since publication of that paper, there has been little movement toward execution of its management strategy.

Not all aspects of the Kundra (2010) plan have been lost. The consolidation of DOD datacenters by 2015 is well underway and the "Cloud First" strategy is gaining momentum across DOD. The U.S. Navy has launched initiatives to place cloud-computing devices on U.S. Navy ships. This initiative initially is only applicable on aircraft carriers because of bandwidth constraints on smaller ships today. Large deck ships may not be the classical datacenter. The amount of data that users generate or use in their daily jobs coupled with the number of applications residing on a single network, lead to the potential of a single purpose datacenter on a ship. Despite this, other aspects of the management strategy are still misaligned, such as the acquisition process and the failure to empower the program manager to field new technologies as part of their existing development. The U.S. Navy could benefit from defining a process similar to the one used by Fujitsu and shown at the high level in Figure 6.

D. SOFTWARE PORTABILITY

Software portability often defined as "portability, in relation to software, is a measure of how easily an application can be transferred from one computer environment to another" (Janssen 2014b). Software portability incorporates general abstraction or virtualization from the logical application and the operating system. Datacenter expansion resulted from having a single hardware platform hosting a single application, co-located with many similar configurations in a single physical location. Applications designed for specific environment, not leveraging virtualization are typically not good candidates for datacenter consolidation. Many applications in use by DOD can operate in a datacenter; however, there frequently are performance issues resulting in usability and

reliability issues. Developing strategies for datacenter consolidation without detailed development guidance and standardization for the program manager to follow can result in poorly designed applications. Software portability was an attempt to help migrate legacy systems developed for specific environment into virtual containers and hosted on common hardware with other similar applications. The promise of software portability continues to lag with DOD applications because many systems require special hardware or software linkages to hardware that do not work in well-virtualized environments. Legacy application constraints not considered during the datacenter consolidation studies, caused delays and difficulties with planned migrations.

Mooney explains that, while most developers agree that portability is a good thing, there is little guidance for the systemic inclusion of portability considerations in the development cycle (Mooney 2004). As with most organically built software, the life cycle of the software often outlives the original expectations. For example, in DOD there are many applications that support critical decision making processes that are 30 years old. As a result, there is always a mix of new software and old hardware or old software and new hardware. No one could predict how dynamic the information technology age would be or how fast it would change. Because of these rapid changes, the move to consolidated datacenters and cloud computing with DOD and other changes in how software is developed must be included in future acquisition strategies. As technology changes, there needs to be a clear path that allows changes in both hardware and software throughout a system's life cycle. This also points to a closer alignment to the business strategies, acquisition strategies, and system level architectures within DOD.

E. OPEN SYSTEMS ARCHITECTURE

Open Systems Architecture (OSA) is a set of standards used in the procurement of Commercial off the Shelf (COTS) products. OSA is a mandate for all development efforts; however, program managers need clear guidance on how to validate and test OSA. In the past, many systems were closed systems, meaning that only the company or government agency that built the system had the knowledge to make changes. The key concept of OSA is rather than building an entire system as one, the system is broken into

smaller logical modules. This provides the ability to upgrade modules individually (Lyle 2013). DOD's biggest challenge with open architecture is taking legacy, proprietary, closed systems and adding open interfaces to them (Serbu 2013). Open standards brings a set of data rights models that ensures that the government owns the data and can use the data as needed to maintain and support the system in the future. As the change from hardware to software intensive systems continues to grow, data rights continues to be an area of great concern to the government. OSA being a standard and not specific to a system integrator or developer helps to better define the use of data rights. Not having government-purposed rights meant that the only company that could modify the source code was the original company that built the software. This, combined with changes in technology, will cause a growth in the Information Technology (IT) budget in order to keep systems current with Information Assurance (IA) and capabilities standards. DOD has established a set of guidance processes and parameters that program managers need to follow in order to meet the intent of the Open Standards policy. Program managers can incorporate the contractual language to ensure that applications and software intensive systems have open interfaces; follow a modular design best practices process; and contracted with appropriate government rights. OSA is as much about the contracting process as it is about the development process. "The challenge as an engineer is to create that architecture so that it allows growth for interoperability as well as system performance" (Lyle 2013).

F. INTEROPERABILITY

Interoperability has its own set of standards and dependencies, similar to OSA. Interoperability definition varies in many ways. The most common definition used in defense as defined by the Joint Interoperability Test Center (JITC 2014):

> The ability of systems, units, or forces to provide data, information, materiel, and services to and accept the same from other systems, units, or forces and to use the data, information, materiel, and services so exchanged to enable them to operate effectively together.

The National Security System (NSS) and Information Technology System (ITS) interoperability requirements includes both the technical exchange of information and the

end-to-end operational effectiveness of that exchanged information as required for mission accomplishment (JITC 2014). Interoperability is a mandated requirement that comes with a cost to the development of any application. Consequentially, all defense systems require testing for interoperability certification with JITC as the key test site. Note that the focus of this section is not to define all the aspects of interoperability, but to show how interoperability can be a powerful tool when following common practices linked with OSA. The JITC definition is very specific for defense systems that includes interoperability and OA. A common term used in the commercial market is Interoperable Open Architecture (IOA). IOA is not yet a commercial standard, Table 1 provides some basic relationships between the two initiatives. Remote Technologies Incorporated (RTI) developed a whitepaper on IOA. They contend that interoperability has been used, abused and confused with many other "ilities" (Interoperable Open Architecture 2012). The table below provides some terms, their technical definitions, and their relation to interoperability in a commercial context.

Table 1. List of Interoperability "Ilities"
(after Interoperable Open Architecture 2012)

Term	Technical Definition	Relation to Interoperability & Commercial Context
Interoperability	The ability of systems, units, or forces to provide services to, and accept services from, other systems, units, or forces, and to use the services so exchanged to enable them to operate effectively together.	A fundamental prerequisite for an open competitive supply chain. Defines an in-service system capability as much as an initial development capability. It enables integrators to connect multiple components developed by different parties without changing them.
Integratabilty	To be able to form, coordinate, or blend into a functioning or unified whole. To incorporate into a larger, functioning or unified whole.	Makes no claims as to the system interoperability. In extremis, any soft-ware system is integratable – at a cost. Does not imply any in-service system attribute.
Replaceability	One thing or person taking the place of another especially as a substitute or successor.	While a replaceable sub-system is an asset, it does not imply that the replaced system enhanced or altered in any way – in fact, it is more likely it has to remain identical in functionality.
Interchangeability	To put each of (two things) in the place of the other, or to be used in place of each other.	An improvement on replaceability because the sub-systems are likely to be able to behave differently based upon the system or sub-system. However, this system context usually has to be pre-determined and fixed before development.

Extensibility	The ability to add new components, subsystems, and capabilities to a system.	There is no limit on the domino effect of change requests needed across the rest of the system to integrate the new sub-system. Nor does it imply that the new system can meaningfully exchange information with any sub-system already in the system.
Componentization	A software package, service or module that encapsulates a set of related functions that communicates via defined interfaces.	A valuable building block in software architectures, its interfaces not necessarily openly defined for interop-erability and still end up delivering stove piped systems.
Modularity	Clarifies the functional blocks of a system, separating capability into modules.	Improves maintainability but makes no claims for interoperability, as interfaces can be closed and proprietary.
Portability	The ability of something, usually a software application, readily moved from one environment to another, usually due to a common platform.	While this aid re-use of the application, it has no association with interoperability of the application with other applications in the environment it moved. It only facilitates integratability with the platform.
Open System	Provides for 'some' level of system capability that exhibits interoperability, portability and use of open standards.	No standard to which the level of openness defined, or interoperability relates.

RTI contends that defense systems should adopt the IOA terminology as the standard for both interoperability and open systems. "By mandating, managing and verifying interoperability the DOD seeks to more closely align defense market with the operations of the open commercial market" (Interoperable Open Architecture 2012).. In general, this makes sense; however, there is a major difference in how the interoperability and open system approaches influence business models. Noted is, RTI as a commercial company, has bias toward the use of open architecture, which may not be in alignment with DOD implementation. The commercial market makes money on being open; however, most integrators working on defense programs do not want open standards and interoperability. The lifeblood for many tier 1 integrators and developers is building closed systems and maintaining them for the life cycle of the program. By enforcing the standards of IOA, the life cycle of application development becomes an open commodity for many program managers. Once open standards and interoperability are achieved, development becomes a commodity that any company or agency can incorporate for the deployment of future capabilities. The best examples are the iOS and Android operating systems, discussed in a later chapter. By creating an open-standards based system and providing guidance on how to develop software allows almost any developer the ability to create compatible applications. Oftentimes, the development of applications in the open systems environment incorporates data sources and information from other systems creating virtual interoperability within the software ecosystem.

G. CHAPTER SUMMARY

Chapter II discusses the defense directives that influence the decisions of program managers in the execution of their product lines. Some directives affect the development while other are more focused on the environments. Other directives pertain only to technology decisions while others affect phases of the product life cycle. The JIE is an example of a life cycle change. It is an enterprise view of how systems will converge in the future. It provides a target architecture and clear goals by fiscal year. JIE provides a phased approach to achieving a sustainable enterprise in the future. What JIE lacks is the "how" for implementation. As with many directives discussed in this chapter, up front and early knowledge of the objectives was good. How to actually implement, and in some

cases, sustain the changes was not as clear. Interoperability and OSA are two directives that do not come with additional dollars. Their focus is on saving money in the future if implemented right.

Datacenter convergence and cloud computing, though separate directives, are closely related. Datacenter convergence aims to save hardware dollars, provide better services to end users, and allow program managers to remove themselves from the constant change of hardware platforms. Cloud computing is not a datacenter, but is heavily dependent on successful convergence of the datacenter. From hardware platforms to new technologies, cloud computing is an abstraction above a datacenter. Cloud computing relies on multiple connected datacenters to provide uninterrupted services to the end user community. In the commercial community, there are multiple cloud environments often overlapping the services they offer their customers.

The Fujitsu case study showed how a commercial company incorporated lessons learned from its own development environments to build a cloud-computing center. Leveraging the internal development challenges, Fujitsu created a commercial market offering and expanded its business portfolio. With more than a $9 million reduction in annual costs and a reduced footprint, Fujitsu realized a major benefit to their cloud computing efforts. Citation Fujitsu used this cloud computing initiative to expand their commercial offering of cloud computing by responding to the needs of their customers. Items like dynamic resource management and automated server deployments meant that Fujitsu could reduce their workforce and increase profit margins.

The Fujitsu case study also re-enforced the notion of a common approach to implementation of new technologies. Chapter II breaks down the steps Fujitsu took with their migration to cloud computing and created a common high-level cloud computing deployment model. A simple model such as this can be the basis for how to implement cloud computing for program managers. Consolidation and virtualization are the key tenets that build the foundation the datacenters need to support future cloud computing initiatives. Standardization is key to the program manager developing or deploying systems or applications into the cloud environment. Life cycle support provides the

sustainment and long-term goals, which often cost more than the initial development of a system or application.

Though the drivers for having the directives may be different, the result is that a solid set of lessons learned can help both the commercial market and DOD program managers alike. Future directives need to have clear guidance on implementation and validation. Leaving implementation to the program manager to figure out results in stovepipe or closed system designs, which then result in systems that do not share data or work well together.

III. INFLUENCES TO DEVELOPMENT

A. INTRODUCTION

This chapter discusses changes to the development process. These changes often cause deviation from the original plan and, in the case of software development or application specific development, changes to the final system that were not part of the original contracted effort. The acquisition process governs all defense systems. DOD spends a lot of money training program managers and leaders in the acquisitions process. From capabilities, to requirements, to development, the process has a number of checks to validate that the program managers and their staff are following the guidance. This chapter reviews some of the key aspects of the guidance provided as the core acquisition process. This chapter will also introduce a high-level reference model. Program managers leverage the acquisition processes to guide them during the product life cycle, but during the development phases, there is little guidance on how to develop the system or application. As discussed in the previous chapter, directives supporting OA or interoperability are required. The program manager needs to determine how to build these features into the system. Use of a common reference model and commercial best practices help a program manager's decision process in the development phase.

B. ACQUISITION PROCESS AND GOVERNANCE

The mandates and strategic goals from senior leadership continue to drive the strategic plans to which program managers need to follow. This has always been part of the acquisition process; however, there is an increasing emphasis on cost savings and alignment to commercial best practices. Strategic plans often provide some level of insight as to what the future systems will need to do from a capabilities perspective. Oftentimes, strategic plans indicate which commercial technologies to target for future implementation. Typically, what is often lacking is a common acquisition approach as to how to acquire an integrated solution. As an example, SOA benefits are dependent on strong governance. Without strong governance, the benefits can easily be undermined, resulting in increased cost and resulting in no value to the development process (Longworth 2005). A key aspect of adopting SOA is that the developers adhere to the

same standards and policies. As an example, Longworth states that based on research done by BAE and InfoWorld, the issue of developer guidance will become more important as more companies try to deploy a SOA strategy across their enterprise. The tools to help enforce SOA policies are just emerging, yet the DOD continues to push SOA as a key enterprise strategy driver for future cloud systems and capabilities. Senior-level strategies should align high-level guidance on how to implement and manage the technologies for implementation (Longworth 2005).

Per the DODINST 5000.02, program managers are required to follow the Joint Capabilities Integration and Development Systems (JCIDS) process. There is a requirement for an alignment between JCIDS (capability requirements and non-materiel solutions), the Defense Acquisition System (DAS) (materiel solution), and the Planning, Programing, Budgeting, and Execution (PPBE) (resources) process (Joint Capabilities Integration and Development System [JCIDS] 2012). These processes work together to ensure that there is a consistent decision-making framework that supports delivering a timely and cost effective solution to the warfighters. Figure 7 below shows the milestone process that each program goes through.

The PPBE helps to establish the strategic goals for future programs and the capabilities they need to deliver to the warfighter. JCIDS supports the process of identifying, validating and prioritizing joint requirements. The DAS take requirements and turns them into capabilities that the warfighters need. Program managers have to follow the DAS when developing products based on the requirements. The DAS specifies the milestones and decisions points before a system can move to the next phase.

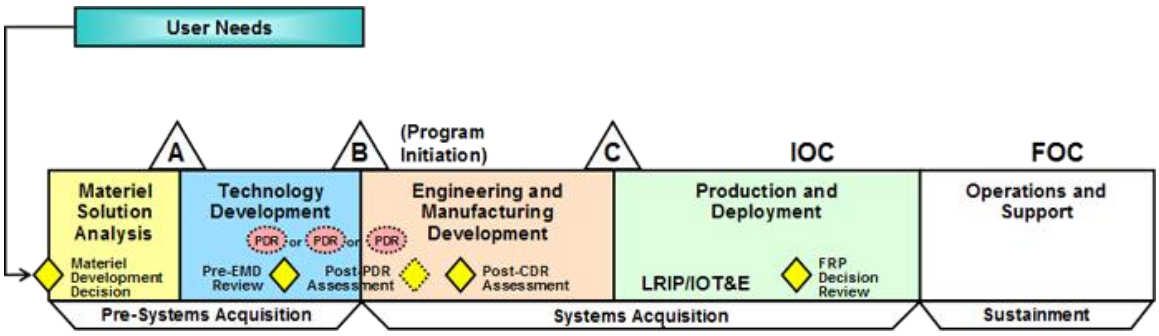

Figure 7. Defense Acquisition Process (from DAS 2015)

Once the acquisition process starts, any changes to the requirements or technology could force the program manager to return to the beginning of the process. When there is a decision to inject a new technology or concept into the process, programs either have to find a way to insert these seamlessly or go back to a previous decision point and re-validate the status of the program. Governance and standards established for the new technologies or strategies will obviously influence development. Ideally, before development starts, defined standards and a validation process would be in place.

The report "Isolating Patterns of Failure in Department of Defense Acquisition" showed that many programs are now more software intensive, which drives the cost and schedule of the programs higher. This may introduce an unexpected dependency on hardware owned by other programs. The article goes on to state that many programs start as part of the same project but end up having competing goals. There are two key challenges: first, development under one program manager with multiple stakeholders (i.e., requirements jumbled into a mixed schedule causing components to compete against each other), and second, a project is aligned to strategic goals at a higher level, where multiple program managers compete for resources on a single hardware platform (i.e., consolidated datacenters) (Brownsword et al. 2013).

The paper defined two critical aspects that affect the acquisition cycle and how program managers try to align their programs to strategic goals: mission and business goals. These goals are not exclusive to each other; often they overlap or lead from one to the next. The mission goal is an expression of some operational objective, focused on what the solution should do or how it should behave. The business goal is an expression is relative to the organizational objective, focused on goals relative to the organization and not specific to the solution. The findings of Brownsword et al. are the result of the SEI's staff work and their relationships with key customers in the defense sector (2013). Aligning goals and objectives is a key aspect of the acquisition process.

Table 2 provides a brief comparison between DOD goals and those of the commercial market. A simple mission goal that starts with a particular focus quickly changes as it goes from leadership to development to deployment. Because of this, defense program managers have to be risk-adverse and often settle for technology and

processes that are older and validated in order to be successful. In contrast, the commercial market accepts risk in hopes of gaining market share and increase profit margins. Timelines are a second area of difference; development takes longer in DOD, whereas the commercial market presses to deliver new products to the market.

Table 2. Goal Comparison

	Defense	***Commercial***
Mission Goal Owner	Senior Leadership	Senior Leadership
Business Goal Owner	PEO or Program Manager	Product Owner
Perspective	High Quality – Complete Testing, based on requirements and capabilities	First to Market – Market Share Focus, first to market goals
Longevity	Long Shelf Life – Upward of 20 Years or more	Until Next Available Version – Months to Years
Variance	Often Multiple Versions at the Same Time	Replaced Quickly and Supported for Finite Period – Typically 5 Years

Commercial markets can set sales objectives early in the process based on market research. DOD focuses on building applications and products established via the requirements process. DOD builds a program schedule based on available dollars and schedules. At the perspective level from the table above is where the commercial market and the DOD have divergence in the development process. DOD products have to follow a rigid schedule established as part of the acquisition process. The next section of the paper explains the requirements process and some of the drivers that influence the development process.

C. REQUIREMENTS

Requirements come in many forms. Most requirements start, as user needs statements or capabilities required to execute a mission. These types of requirements are

easy to define and quantify costs for inclusion in an existing system or a new system. Unforeseen derived requirements may change schedules or increase costs, making it difficult for a program manager to field a system. Derived requirements come from many different mandates, policies, or strategies; they can also be the glue that turns an initial requirement into a capability. As discussed in previous chapters, there are many types of unfunded mandates, policies, and directives, most related to architecture strategies or technologies used in the commercial marketplace. During the development phase, changes to requirements or capabilities become derived requirements. An example of such a derived requirement is security, further discussed in the next section. Security is an implied requirement that does not correlate to the mission of the system as defined by the JCIDS process; however, it is an imperative element of system performance. Security is a mandatory requirement, not typically associated with user needs or capabilities, but heavily governed by mandates and policies. SOA is another example of a derived requirement. It is an implementation strategy based on technology, thus becoming a requirement. SOA enables systems to better communicate and share data. SOA is an approach for organizing and using services to enable interoperability between data assets, applications, and users (Shea 2009). SOA as an enabler, but not a hard requirement. As a derived requirement, program managers need to determine the value of an implementation approach to the development process.

As required in Title 10 of the United States Code, the program manager signs a document that states that for the life of the system or application, they will maintain all aspects of that system or application. According to DODI 5000.02, "life-cycle sustainment planning and execution seamlessly span a system's life cycle, from material solution to disposal" (DAS 2015). Known as the Life Cycle Sustainment Plan (LCSP), LCSP defines a product's support availability, reliability, and affordability for the product's life cycle. Program managers need to present a milestone B brief to the Milestone Decision Authority (MDA), which contains the LCSP. Once the system or application is delivered, the funding profile changes to sustainment funding. Development of a new capability or insertion of new technologies is difficult once the program is in sustainment.

The acquisition, development, and life cycle support elements are the foundation that all program managers manage. From the conception of a need to the development, deployment, and eventual retirement of the system, program managers own the cost of the program. There are activities that program managers have to coordinate throughout the program's life cycle. There a number of DOD specific processes, like JCIDS and JROC to guide the requirements process. Lacking is the "how" in the development cycle. This lack can be addressed by the use of commercial standards applicable to the use the commercial technologies. Figure 8 is a simplified version of the process. It lays out some key aspects of each of the three major phases for program managers. This is the same figure shown in Chapter II to describe the acquisition process at a high level.

Figure 8. Simplified Program Life Cycle Process (after DAS 2015)

The acquisition phase of a program life cycle includes policies and guidance to which the program managers need to adhere. Most of the guidance found in the DODI 5000.02 documents and the DAP focus on the initial phase, Acquisition. The Life cycle phase comes after development is completed and the product is in use. Program managers are left to define the development of the application based on either industry standards or limited DOD guidance. Program managers have to report on the status of the development phase to senior leadership. Program managers explain how they are complying with policies or directives to the MDA based on the acquisition process milestones. Chapter IV provides some examples of how industry handles development guidance. Because DOD has systems that remain in use for many years, DOD needs to provide more guidance to the program managers on the development process.

D. CYBERSECURITY

One of the more difficult aspects of any system is making it secure. Security is an ever-changing posture for any system. The DOD Cybersecurity Policy Chart captures the tremendous breadth of the applicable policies in an organizational construct (Cyber Security and Information Systems Information Analysis Center [CSIAC] 2013). The requirements to ensure compliance often changes from the initial start of the acquisition process to retirement. The Cybersecurity Chart (CSIAC 2013) defines the goals as:

(1) Organize for unity of purpose and speed of action;

(2) Enable mission-driven access to information and services;

(3) Anticipate and prevent successful attacks on data and networks; and

(4) Prepare for and operate through cyber degradation or attack.

Dittmer explains the organization and purpose to help the Information Assurance (IA) professionals (2010). Security as a critical aspect of any system today needs to follow very strict guidelines and policies. Building a system that is both secure and modular in design from a security perspective is very difficult. Security is a functional requirement; however, it is often considered a derived requirement and at the bottom end of the funding priorities (Dittmer 2010).

Within the acquisition process, there are references to IA strategies and guidance. IA is about risk of system exploitation. As outlined in the Cybersecurity Policy chart (CSIAC 2013), an IA Strategy is a standalone document that the program manager and program office use to organize and coordinate its approach to identifying and satisfying IA requirements consistent with DOD policy. IA is constantly changing; our adversaries use vulnerabilities to access data that could help them gain a strategic or tactical advantage in a conflict. IA standards for protecting the data and systems is constantly changing because of technological advancements (Dittmer 2010). Often, as the tools for evaluating the posture of a system change, the risks to the program change. Program managers have to build a system that is secure, remains on schedule, and within cost.

E. SOFTWARE DEVELOPMENT

A 1980 study by the Electronic Industry Associates (EIA) on DOD computing stated that if "if software costs increase one order of magnitude every 10 years, then by the year 2015, software will consume the entire defense budget" (McDonald 2010 32). The point of the study was to expose the impending problem state of software development within DOD. DOD standards for software development do not imply that there will be cost savings, common use of the standards, or even acceptance of the standards. In his paper, McDonald brings to light the many failed attempts to control the commercial software industry by imposing standards mandated for programs within DOD. One such example is the B-52 bomber. Prior to its first major software upgrade and after more than 30 years of use, the aircraft had only six major critical anomalies. During the testing of the development program, the discovery of 34 mission critical anomalies resulted; 14 identified as new avionics hardware and 20 identified as software. Subsequently, programs were seeing similar issues and defects. It was common knowledge within DOD that software was driving the cost of programs to be cancelled or significantly reduced (McDonald 2010, 34).

Starting in the 1960s, the field of software engineering began to emerge. In 1968, NATO held the first ever software engineering conference. One of its focus areas was to investigate the application of "engineering principles to computer programming."

Additionally, the conference defined new processes and ways to manage software; in essence, it established standards for software development (McDonald 2010, 34).

The Navy took the lead in establishing software standards under the leadership of the Naval Material Command (NMC), the logistics arm of the U.S. Navy. NMC wrote the first draft of the proposed standards, released in 1978 as the first military standard for software development (MIL-STD-1679 Weapons System Software Development) across DOD and introduced for use in military contracts. MIL-STD-1679 followed commercial industry best practices at the time. MIL-STD-1679 also provided a set of detailed coding guidelines. Many of the programs at the time believed the standard was very constrictive to their innovative abilities (McDonald 2010 34). As with many of the standards in DOD, where there is a commercially acceptable best practice, industry typically will resist the adoption of DOD standards. There were different efforts in which DOD wanted to see more rigor and standardization across the software development communities. The Joint Logistics Commanders (JLC) was working on developing their own set of standards as the Navy approved the MIL-STD-1679 in 1978 (McDonald 2010, 38). JLC focused on how to align the acquisition process to the software development standards. They wanted language in each contract to ensure adherence to certain software guidelines. The subsequent development of DOD-STD-2167 aligned the software life cycle to the acquisition process.

DOD-STD-2167 included wording and direction that the development cycle shall follow the waterfall methodology. The standard also provided the ability to customize programs and in fact, had a 13-page section for how to tailor the standard. Critics continued to complain about the rigor and imposed limitations of the standard. JLC argued that the entire standard supported tailoring to fit the needs of the program.

F. CHAPTER SUMMARY

Chapter III introduces the acquisition cycle as a complex process often adding to the challenges that a program manager needs to work through while trying to build a system. Requirements flow from JROC, then validated through the JCIDS process, and finally delivered to the program managers as the capability for development. Program

managers are supposed to follow the acquisition process for development of the products; however, specific guidance for how to develop a product is up to the program manager. Requirements are a key aspect of the acquisition cycle and the development cycle. In the acquisition cycle, capabilities become requirements. Requirements define the set of functions, which in turn define the basis for development. These allocated functions provides the scope of the development effort. This includes cost, schedule and performance. During the development phase, most architecture and framework requirements become derived requirements. Derived requirements often come from directives or policy changes that have nothing to do with the capabilities desired. Security, the hosting environment, or technology insertion such as cloud computing, could be driving factors in getting the capabilities to the user community. Typically, these requirements are not included in the price estimate during the program-planning phase. Cost, schedule, and performance changes influence the ability of a program manager to execute the planned development. They define the trade space that ultimately define the product.

Over the years, DOD has tried to influence the development phase for many programs. Software development has seen many changes over the years as DOD has tried to lock down the standards for all programs including dictating methodology and software languages. Commercial software efforts are more open in that guidance typically comes in the form of lessons learned and industry-based standards. With the direction to use more COTS products, program managers have had to define the development environments that best meet their needs. This further compounds a program manager's confusion when they enter the development phase since there is no specific guidance for the development of a system. This will be the focus of the next chapter; it will also answer the question, how can DOD provide clear guidance based on industry standards without limiting the innovation of the development community?

IV. COMMERCIAL APPLICATIONS DEVELOPMENT

A. INTRODUCTION

This chapter discusses examples of how the commercial market place uses the latest technology to field new applications. Technologies referenced in this chapter are the same products targeted for use within DOD. This chapter will explore the guidance and lessons learned from the commercial market. This chapter will show a comparison of the challenges associated with following commercial standards in DOD as compared to commercial standards in the open market for consumer usage.

The commercial market defines the approach to their development and then provides documents so that others using the methodology has a level playing field. In contrast, DOD has to be selective in the technology and methodology they use to build the latest applications. In the commercial market, an 80% solution is fine to go to market. The commercial goal is to be first to market and leverage the users as testers to finish the products. It is not that the commercial market purposefully sells products that are not complete, but having a 100% complete product requires extensive testing and often much longer than the company can wait. DOD requires exhaustive testing and validation to meet specific requirements before product deployment. Applications developed for DOD usage often contain specific user needs; if an application does not meet these needs, then the product is not usable. User requirements for commercial market place can be more related to ‘nice-to-haves’, whereas in DOD they are operational capabilities of the users. This may not seem like a major departure from a definition perspective, but as discussed in the previous chapter, the acquisition process requires a rigid validation process. Most commercial companies follow a standard framework and architecture for all development efforts. The methodology may be specific to a company or technology, however, the methodology documentation design for repeatability. DOD supports many user communities and has many different development communities. The commercial market generally singles up on a single common framework and architecture approach based on product requirements. DOD consists of various mix-and-matched frameworks and architectures. In DOD, prime contractors drive the development for each product line.

DOD does not mandate any specific architectural approaches or specific frameworks that developers shall follow, only high-level guidance, which all outlined at time of contract award, is provided. This adds to the confusion caused by interoperability. Joint Interoperability Technical Command (JITC) is responsible for validating interoperability requirements. Every product that crosses organizational lines has testing requirements for interoperability.

In light of the preceding comments, this chapter will explore the use of software development kits (SDKs) as a strategy for integrating the DOD environment with the commercial marketplace. In the end, this chapter will demonstrate that a common approach to development with common guidance will prove that DOD can do a better job at building applications with technologies available today. Some additional restrictions like security and open architecture are required as discussed in previous chapters. By defining a common framework or methodology to software development, additional restrictions or guidance fit into the process.

B. SOFTWARE DEVELOPMENT KITS

With the evolution of the personal computer, the market has become one where every phone or tablet with an Internet connection is more powerful than a personal computer developed a few years ago. Software development kits provide the developer community everything they need to know to develop applications quickly. A user with a good idea can log onto the web and follow a common process to build an application. Google (Android) and Apple (iOS) provide clear steps that make it possible for any user to build an application. There are even third party development kits that allow for the porting of an application built on one framework to different frameworks. Both Google and Apple have spent many years perfecting their own processes and guides to ensure users have the same level of knowledge as professional services in building applications in their individual environments.

In contrast, DOD hires prime contractors for most of the software development. Many of these prime contractors only build applications to very rigid sets of requirements on very specific frameworks. The capabilities desired and the legacy processes that the

acquisition process follows limits innovation. Oftentimes, technology that is cutting-edge at the start of a development cycle is nearing obsolescence before the development is completed. SDKs developed by prime contractors or leveraged from older contracts, often contain little to no modernization. Once the development phase is completed, the testing phase as defined by the acquisition process starts. In many cases, the software is still not in the hands of the users for some period to come. Lastly, once deployed, it often takes many years to replace the software that was near its end-of-life before fielding.

Webopedia.com defines an SDK as a programming package that enables a programmer to develop and application for a specific platform (http://www.webopedia .com/TERM/S/SDK.html). In general, DOD is platform-agnostic by nature. Depending on the requirements and the system capabilities, the platform requirements could change. For example, a software system supporting a real-time system may have different specifications than an application that tracks the number of candy bars sold in a ship's store. These applications may not use the same software platform. However, if DOD had a common methodology for development, more processes that are common could influence the development of software. Figure 9, introduced earlier in the paper is an example of the simplified program life cycle chart. There are three phases; the first, detailed in Chapter III, the acquisition phase. The focus of this chapter is the development phase. The development phase is often defined by the contract type or the capabilities desired by the user community. A high-level software developer's guide at the DOD level would ensure that a common development methodology used across all like development efforts. No single SDK will fit all cases, but lack of any guidance is not the answer either.

Figure 9. Simplified Program Life Cycle Process

C. COMMON ELEMENTS OF THE DEVELOPMENT PROCESS

Most SDKs have a common set of steps that are required. Some of the most popular SDKs in the commercial market today are those used to develop applications for Google's Android and Apple's iOS devices. Both these companies have SDKs online as well as a number of resources available at local bookstores and online bookstores. There are many examples of developers building products for both product lines. In 2013, Henneke built the same application for both Android and iOS devices. His methodology consisted of lessons learned, tools, design, data and storage management, testing, and security (2013). The below list of steps is specific to his efforts developing on Android and iOS platforms, but also provides insight into a common approach.

1. ***Lessons Learned***: A developer has a number of resources available to them before starting any new project. A key resource is the published lesson learned. Some of these come from other developers, some from the software manufacture, and some from experience. No matter the source of the information, lessons learned can help jump start a development effort.

2. ***Tools***: A common set of tools that can support the developers with coding and debugging the code. Tools become important to ensure that good quality code is used and that the code meets certain requirements for security and business processes. Some platforms and software development environments come with specific tools that help to improve the performance of the software. If the developer is new or inexperienced in a specific environment or with a specific operating system, the documentation will be the initial starting point before building the application or system.

3. ***Design***: The design influences how an application will act on certain devices or hardware platforms. With Android, it can be very complicated, as different manufacturers make the devices. There are also a number of older systems still in use, so backwards compatibility is a consideration. Apple is a little easier, as they limit the number devices that an application has to support. They also do all the manufacturing specifications for all their devices. Therefore, there are common aspects to all Apple products. Developers have to consider all factors when developing a new application or system that will host other applications. The days of designing for a specific hardware platform or singular environment are past.

4. ***Data Management and Storage Management***: Many mobile devices are not large enough to maintain huge amounts of data locally. Data becomes stale; having a well-defined data strategy before starting to build an application or system level set of applications is important. There are tools that both Android and Apple use, but the developer still needs to define the type of data, where will it be stored, refresh cycles, and accessibility with and without network connectivity. A design consideration for data management is how the operating system handles file management. Considerations of the design can affect the data strategy and storage management of an application. Data requirements and applications to access the data generated have a direct impact on design consideration. Interoperability becomes important in the retrieval and exposing of data from the application or one system to another. All these decisions have direct impact on fixed storage like on a hard drive or on temporary storage like Random Access Memory (RAM).

5. ***Testing***: The framework and SDKs need to provide clear guidance and information on how to define the testing environment and parameters for which the application needs to accommodate. Android and Apple have

spent a lot of time and money in helping developers overcome this obstacle. There are also a number of third party tools available to the developers to help debug the software code and test the application multiple times before deploying the code. There is a direct correlation between the tools and the testing prior to deployment of any application. Both Android and Apple also have a final check and review that are required before application deployment to their application stores.

6. ***Security***: With the growing threat to personal information and the amount of information now being stored on mobile devices such as phones and tablets, security has taken on a much larger role for the developers. Developers now have to have specific validations before applications published for certain environments. To protect the owners of mobile devices, manufacturers of the hardware devices, and the software developers, security is an enterprise solution. Cybersecurity is more important than ever before. Previously left until the end of the development process, security is now taking a front seat and designed in the front end and throughout the development process.

Henneke's lessons learned points to the benefit of a common approach to development. In his example of building a common application in two different environments, he showed that by using the same methodology, he was able to do a simple comparison of his experience. A lesson learned for DOD development efforts, the development of a common repository from other efforts and simple to follow processes provided as a reference to all development efforts. Using a common methodology is beneficial in ensuring that no matter the platform selected, the basic processes followed within the development process. By ensuring the same processes, common testing and validation approaches are standard.

D. COMMERCIAL DEVELOPMENT FOR MOBILE DEVICES

A core concept used by both Apple and Google is that all software development tools are available and simple, easy to follow steps are provided. From concept to release, various resources are available to ensure that the developer understands the process. A number of tools provided to the developer to ensure that the application works within the environments. Some tools for checking quality are depending on the environment selected, but quality is not the concern of the Google or Apple as much as ensuring the

integrity of the environment. Regulation of the environment ensures all completed code passes testing by the host system prior to deployment.

Application design follows a standard process regardless of whether developed using the Google or Apple environment. Both Google and Apple provide all the details for how applications work within their individual environments. Clear, easy to follow steps are provided to help the developer take their concept through the design phase. One of the biggest challenges with any application is defining a data strategy. Most applications rely on data and information from other applications or data stores. Defining data retrieval and storing it on the local device, then displaying to the user can be the strength of an application. Developers must understand the connections and services offered by the environment in order to map the application features to the data sources. Many users determine the applicability of an application by the source of the data and the speed at which it works. Google and Apple have built in tools within the SDKs to ensure that the developers can make the best use of the services they offer in their environments. Henneke touches on the ease of use during his experiment to develop a single application for both environments. He spent some time before actually starting his development learning the different environments (2013).

The following paragraphs and figures provide some general concepts of the processes and the environments used by Google and Apple. These are examples, which show many common aspects to development processes. Though these two environments use different software code bases, have different security and integration needs, and are not interoperable, the concept of development methodology is very similar. As shown with the paper, a single application can be built in either environment and have the same operational capabilities on either environment.

1. Apple Development Environment

Apple provides a straightforward development approach for applications on iOS. Figure 10 shows how starting with a concept, often on a piece of paper or even a napkin, the developer can then progress through an easy to follow website that will guide them

through the development process (Apple 2013). Apple provides a three-step process: Structuring the App, Implementing the App, and Next Steps.

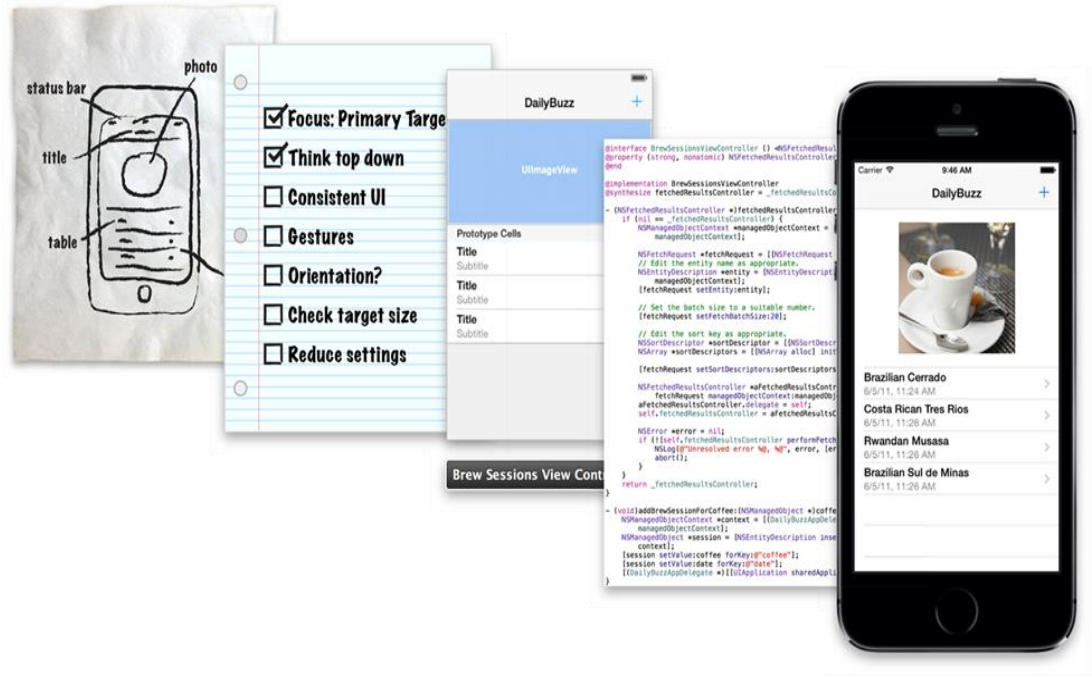

Figure 10. From Paper to Application (from Apple 2013.)

Apple's development website provides most of the tools and design methodology needed for the basic applications. The more complex the application, the more that external resources will be needed. If the developer has experience, then some of the steps may seem to be redundant or unnecessary.

Apple's SDK provides background on how the different layers of the iOS operate. At the highest layer, iOS act as an intermediary between the underlying hardware and the applications (Apple 2013). Applications communicate with the hardware through well-defined system interfaces. Figure 11 shows the four layers of iOS. The lower layers are the fundamental layers or core services. The upper layers provide more sophisticated services and technologies. A key service of iOS is that many of the services are already built and re-usable.

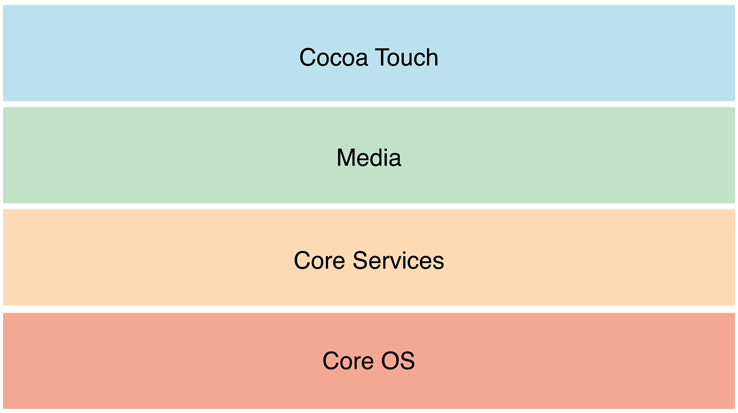

Figure 11. Layers of iOS (from Apple 2013)

The iOS SDK provides all the tools and interfaces needed to develop, install, run and test native applications on iOS devices. The developer library contains the API reference, programming guides, release notes, tech notes, sample code, and many other resources to help the developer through the process.

Apple provides a good example of standard guidance used to help the developer. By providing these resources to the development community, more developers are willing to use the frameworks to build applications for Apple devices. It is to the advantage of Apple to provide clear direction; this helps to increase the number of applications hosted on Apple's environment, which increases the number of applications available to users, which in turn increases the number of users.

2. Android Development Process

Google calls their mobile OS Android. The Android development workflow is straightforward and includes simple principles to keep the developer on track (Android 2014). At the highest level, Android tries to keep the process simple: Design, Develop, and Distribute. Each of these processes contains all the instructions necessary to develop applications for the environment. These tabs can be found at the top of the page and allow the developers to jump through the process or search for helpful hints. The Android approach consists of four steps: setup, development, debugging and testing, and publishing. Figure 12 depicts the steps and flow to guide the developer through the process. All of the tools and library information is readily available from this single website.

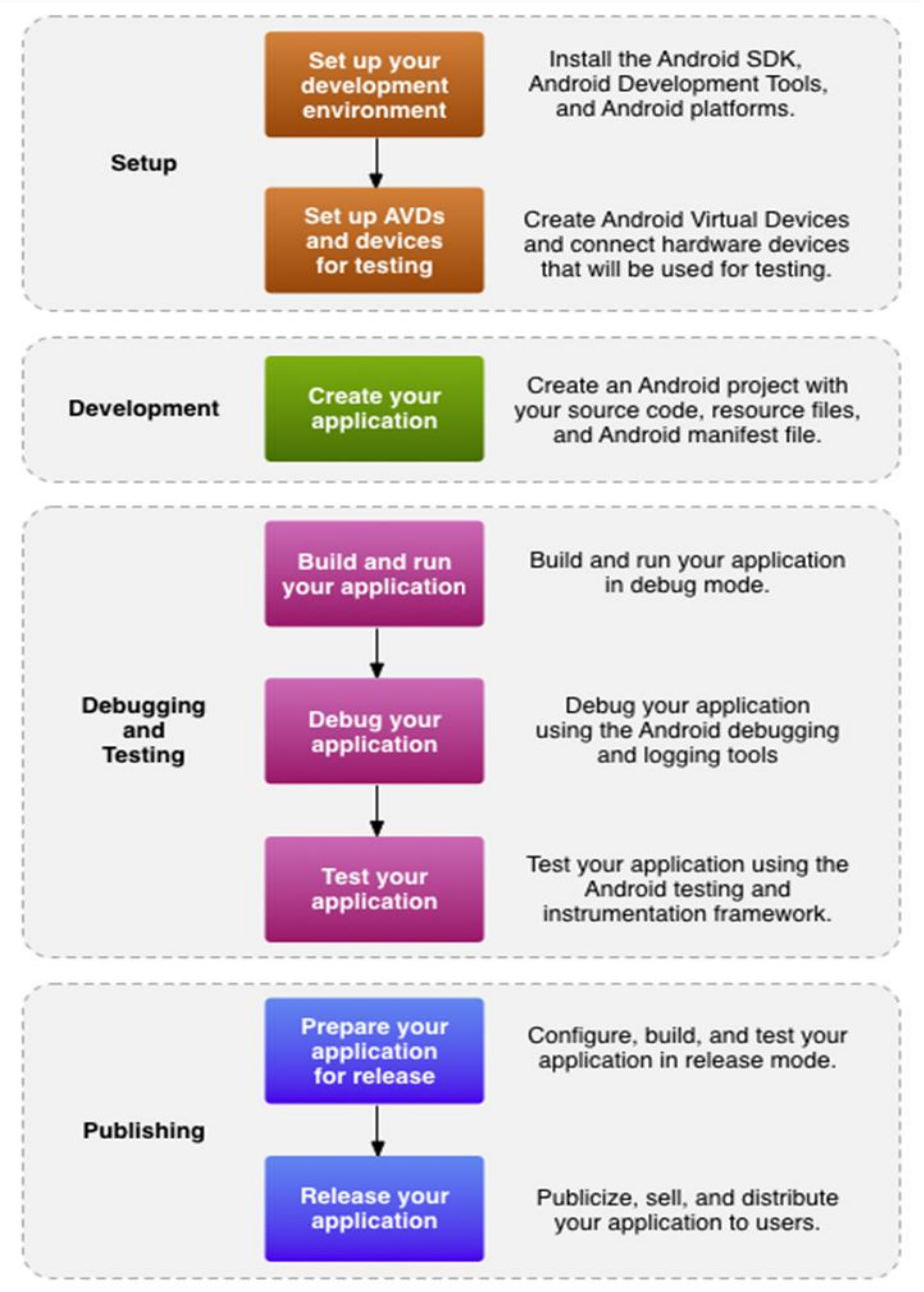

Figure 12. Android Development Process Flow Chart (from Android 2014)

Android provides helpful training and detailed instruction for each phase or step of the process. Development is only a small part of the process for Android. Android provides a lot of tools and help in debugging the software before allowing testing of the application. Emulation tools help to debug the software before attempting to deploy the application.

Both examples are very straightforward and, from an industry perspective, are the leaders in the industry for ease of allowing non-programming types the ability to quickly take a concept and turn it into an application. Figure 13 shows a snapshot of the high-level processes. The names are different, but when you read the steps and follow the processes, they are very similar. Android tends to be a little more predictive in their process, but both follow similar steps. Both Apple and Google have spent many years updating and modifying the underlying framework and software development guide so that anybody can use it. They also provide a plethora of resources to aid in the development of applications.

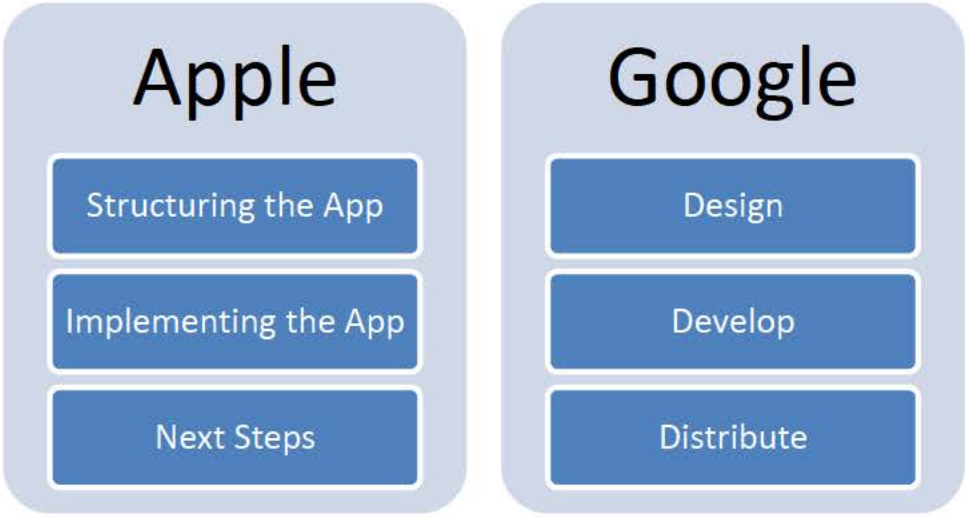

Figure 13. Comparison of Development Processes

Doing a comparison of the operating systems (iOS and Android), there are some differences, as discussed in a paper by Jayaraman (2013) that explores both development approaches. Though the iOS approach is slightly different as it only provides open interfaces, the Android approach provides both open code and open interfaces. The advantage to this is the openness to allow for innovation within the platform (Jayaraman 2013). Because of this openness, both iOS and Android rely heavily on the development of third-party developers. This aspect of development encourages the innovations. Over the years, what has happened is that software platforms have transformed from a platform-centric perspective to an ecosystem. In the case of iOS and Android, the

ecosystem is the mobile environment. A major change in using an ecosystem approach to development is in the collaboration of the development community. In a platform-based approach that is closed and where the development is only focused on a specific capability, there is little need for collaboration or exchange of ideas. With the open market as utilized by iOS and Android, the real value to the end users comes from the exchange of ideas and sharing of information within the community. The other aspect with both iOS and Android is the number of devices that these ecosystems have attracted. The mobile software ecosystem is as much about the increase in the number of devices and users as it is about the ability to add new applications to the platform quickly (Jayaraman 2013).

E. APPLICATION DEPLOYMENT

Deployment has changed with the software ecosystem of iOS and Android. Once built, the key is to ensure that the applications are available to those who either need them or want them. From the perspective of a company that wants to build custom applications specific to the operations of their organization, they need a common way to deploy rapidly at a reduced cost. Creating an internal application store and allowing only employees to access it allows the company to push out security updates, custom applications, and even manage the devices from a central location (Marko 2013). Like the software development environment, the deployment of applications into a common environment is critical to the success of the enterprise. Building a common environment is the first step; having the applications developed for that environment is next. However, neither of these would be of value if there were not a common way to deploy, track, and monitor the applications. An app store specific to the needs of the enterprise is critical to ensure the existence proposed cloud computing environment and/or hosting of common applications as mandated in the strategic plans.

From the software ecosystem to the deployment of new applications, the SDK is key to ensuring consistency in the process. From the perspective of policy and guidance, both iOS and Android defined the ecosystem for both development and deployment. The commercial market place continues to expand, and the use of a commercial or private

application store allows for continued innovation. There are numerous lessons that DOD can learn from these types of ecosystems, but this does not mean that DOD should merely copy this. A top down view to include instruction on to meet objectives provided as guidance to program managers help to ensure commonality and objectiveness.

F. DOD GUIDANCE

DOD does not leverage a common framework of specific guidance for software development of implementation of commercial products today. In a paper from 1994, some of the issues program managers faced with the development of software intensive system highlighted challenges of the development efforts. Many of these same issues face program managers today. The paper stated, "In the last 20 years, DOD has been increasingly criticized about its ability to manage the acquisition of automated defense systems (Shebalin 1994). At the time, the defense industry influenced many of the major software processes and languages used in the commercial marketplace. The commercial market produces more innovative software intensive systems than DOD. DOD has adopted more of a follow-industry-standards approach to the development of new systems, often without any forethought or guidance provided to the program managers. Shebalin (1994) pointed to the use of Military Standards (MILSTDs) for bounding the problem. System development typically falls to the purview of the systems engineer with direction and guidance from the program managers (Shebalin 1994). Oftentimes, there is little guidance or standardization across the DOD enterprise as to how the systems engineer performs their jobs. Software development is one area that can leverage from the commercial market. DOD does not follow a standard process or framework for software development. Implementation of commercial technology typically falls to the systems engineer and is dependent on the environment.

By providing a common guidance and not just industry standards, DOD will ensure that the objectives of the technology or software ecosystem are common across all development environments. By providing a common set of objectives, high-level view of the technology implemented, and considerations for deployment, DOD can achieve consistency in development and interoperability. Table 3 provides a framework that if

defined for new technologies can help ensure that the program manager has a clear understanding of the technology. Additionally, this information will provide a level set of information that the systems engineer can leverage while designing this technology or process into development plans. Often, senior leadership dictates new or cutting-edge technologies injected into the middle of development cycle. The information found in the standard guidance proposed will help the program manager to determine the best time and means to implement.

Table 3. Standard Guidance Approach

I. **Standard Guidance**

II. **Objective of Technology or Software Guidance**

- A. Why this technology
- B. Expected outcome/usages
- C. Importance to DOD
- D. Senior leadership guidance for usage

III. **Industry Guidance**

- A. Design considerations
- B. Commercial software development kit
- C. Commercial uses
- D. Implementation guidance

IV. **Lessons Learned**

- A. Information from company/commercial sources
- B. Examples of how technology is used
- C. Information from other adopters

V. **Security Considerations**

A. Commercial security considerations

B. DOD requirement considerations

C. Security implementation considerations

VI. **DOD Unique Guidance**

A. DOD design guidance

B. Guidance consideration for DOD implementation

C. Environmental considerations

VII. **Tools/Testing**

A. Development tools (if available)

B. Modeling tools (if available)

C. Commercial validation tools

D. DOD testing requirements

E. Deployment validation testing criteria

Though this seems to be a simplistic approach, short of any guidance, engineers will continue to build and field products that only meet the needs of the program manager for a specific capability. Interoperability, open standards, and security are just some of the driving factors that influence the approach that program managers must consider during the development cycle of a program. Without specific leadership oversight and guidance, the objectives of injecting technology or methodology can be lost on the program manager. This guidance points the program manager and systems engineer toward a common set of goals.

G. CHAPTER SUMMARY

The available software development environments that are easily accessible via the Internet govern Android and iOS development. No matter the development effort, a common set of instructions guides an experienced developer or a new developer through

the process. A developer needs only to have an idea and some basic understanding of how they would like the software to work. No matter which platform selected, there are some common elements, tools to help the developer walk through the process, testing tools to ensure that the application will work, and data manager tools to ensure that the required data is easy to access and manipulate. Many of the tools and guidance found in the SDK that both companies offer are openly available via the Internet or in books purchased online or from local bookstores.

The DOD can learn from how the commercial market defines the development environment. A common set of tools and frameworks provide standardization for developers. The following list is a summary of the guidance provided earlier in this chapter:

1. Following common steps
2. Providing clear guidance
3. Using libraries of application programming interfaces (API)
4. Using common set of services
5. Using re-useable objects
6. Use of Common test strategies

DOD leverages many commercial standards in the development of applications. Oftentimes, the program managers are not provided any directions on how to implement the technologies. DOD needs to establish a set of common guidelines for development. The guidelines need to bind the software development ecosystem specifically to the architecture and system frameworks.

V. SUMMARY, CONCLUSION, RECOMMENDATIONS

A. INTRODUCTION

The DOD continues to implement new technologies and struggles to adopt commercial processes for implementation. Program managers are often under great pressure to use the newest and greatest technologies found in the commercial market. Injecting new technology or a methodology into a development cycle increases the technical risk to program. Senior leadership in DOD continues to mandate the use of certain technologies or implementation mechanisms without consideration of the acquisition cycle. Senior leadership challenges program managers to think outside of the box when incorporating the latest and greatest technologies. Cost, schedule, and performance are the standard metrics used to indicate the success of a program manager. Understanding that one can only achieve two of the three. Program managers require usable guidance to achieve success in the technology insertion process. Technology insertion often influences all cost, schedule and performance. As it stands currently, there is no point in the acquisition process where new technologies introduction does not directly affect requirements.

Senior leadership in DOD continues to push the commercial marketplace paradigm; however, in the commercial marketplace, if a technology or implementation methodology does not work, the company loses money. In the DOD environment, if the use of a certain technology does not work, it costs millions of dollars to re-baseline the system. The commercial world provides a number of lessons learned that could help ensure success for government development. Incorporating lessons learned and providing a clear set of policies and guidance documents will better guide program managers in aligning to DOD initiatives.

B. RESPONSE TO THESIS QUESTION

This thesis outlined existing processes that are required for any development effort. This thesis also showed examples of how the commercial market uses a common approach to software development and technology insertion. A single industry-wide

acquisition process does not regulate the commercial market or methodology. DOD acquisition is a regulated process that program managers have to follow in order to meet their milestone objectives. Senior leadership holds program managers accountable to control cost, schedule, and performance in the execution of their program. Senior leadership in DOD often interjects new requirements, which have a direct impact on the execution of the existing program. The JROC process ensures that capabilities needed by the user community meet the user's needs. There is no analogous process for derived requirements. There needs to be a process to ensure that new technologies and or strategies injected into the development cycle enhance the validated requirements. Many of the technologies and initiatives injected into the process follow commercial trends. The commercial market does not have the same 100% testing rigor that DOD has for implementing capabilities. For the commercial market, being first to market is more important that an error free system. Senior leadership wanting to be more like the commercial marketplace should also provide guidance and oversight for program managers to implement emergent changes to existing and future systems.

C. OVERALL SUMMARY

In Chapter I, this thesis introduced some of the background on many of the issues that influence the acquisition of programs in DOD today. Additionally, Chapter I introduced many of the initiatives aligning the DOD infrastructure to the commercial market. Chapter I prepared the reader for the some of the research needed to show the difference between how the commercial market incorporates technology through implementation and deployment methodologies. Many of the challenges revolve around the implementation of a defined software ecosystem and software architecture. DOD does not have the luxury of building a system based on a single senior-level strategy or goal. Each service and agency within the DOD has their own goals and objectives for the application they need to meet their mission. Oftentimes, the only thing they have in common is a loosely coupled enterprise-level architecture. Chapter I identified how loosely coupled architecture and software ecosystems influenced by commercial technologies, may or may not be the right solution for DOD systems.

Chapter II focused on the commercial technologies and the challenges that program mangers face when senior leadership attempts to direct changes to programs already deployed or in the middle of development. Incorporation of commercial technologies at the beginning of a development cycle with clear guidance will help ensure successful development efforts. Chapter II explores commercial standards and guidance. The business drivers are different between the commercial market and DOD, which in turn influences the decision process. DOD development focuses on processes and regulations to ensure that system sustainment is maintained for the life cycle of the program, which is often more than 20 years.

Chapter III reviewed many of these processes and regulations. The acquisition process defines all the milestones that a program has to go through. Each step of the process includes many reviews by senior leadership to ensure the program managers are on track. A program manager manages cost, schedule, and performance in the development of products. It is very difficult to maintain these three key measures when the leadership is forcing them to change technologies and strategies mid-stream.

Chapter IV looked at how the commercial market uses an SDK as a development framework for OS-specific applications used on a variety of platforms. A common complaint from within DOD is how the commercial market appears to make it easy to deploy new applications and capabilities. Chapter IV explores how the commercial market does this through governance and policies. The commercial market established a set of guidance documents and the tools to validate the products before deployment. Commercial companies maintain close control of the framework and architecture; however, they provide developers all the tools needed to build applications on the framework. Android and iOS (Apple) operating systems provide clear guidance to design, build, and deploy applications in their software environments. If DOD is to adopt commercial processes, it needs to start by establishing a common set of development methodologies and a common framework. These companies have developed a clear set of rules that every developer has to follow. More importantly, they have made it very easy to use and follow. By defining the environment and easy-to-use processes, the companies can better manage the applications behavior within the environment.

Chapter IV also provided a proposed simplified DOD standardized guide. This standard guide supports software, hardware, and technological approaches. A standard guide provides a common set of rules that program managers and systems engineers can use to ensure alignment to senior leadership strategic goals. DOD can learn this from the commercial market; however, they need to provide context as to how these lessons influence the DOD architectures. Taking lessons from the commercial market is only a starting place; DOD has unique requirements like security and interoperability that require specific considerations for implementation. By establishing a common framework and methodology, DOD can ensure a more sustainable environment across DOD architectures.

D. CONCLUSION

As shown in the proceeding sections, clear policies and guidance for how to align "enterprise level solutions" has gained limited traction in DOD. Program managers use cost, schedule, and performance as the driving factors of success. Adding or changing the alignment of the development effort is discouraged since it will increase (in most cases) cost and extend the schedule. Further, performance becomes the variable that program managers use for fixed cost and schedule objectives. There could be advantages in that the performance of the system may improve, but clear analysis to support such transitions is costly and could affect the cost and schedule of the program. However, given the current acquisition process, most program managers will take the path of least resistance and focus on the execution based on the lowest technical risk path approach.

Therefore, senior leadership needs to provide a clear set of objectives for any new technology or change to development. Without a common framework to work from, program managers will continue to build systems using legacy approaches. Table 3 is only a small sample of the common guidance that could help drive technical solutions to a common methodology. Importantly, there is a technical risk of not having a clear set of guidance documents today, which affects future program sustainability and affordability. In contrast, the commercial market provides a good set of lessons learned for how development can work, but the commercial market and DOD are vastly different in terms

of sustainability and affordability goals. Accordingly, considerations in methodology, framework, and implementation need to occur early in the development process. All development plans should start with a common set of objectives to ensure that program managers have a clear understanding of senior leadership objectives. In the final analysis, technical issues are easy to overcome if one starts with a clear set of technical objectives.

E. FUTURE WORK

A potential topic for future work focused on the JROC processes with relationship to the commercial best practices and changes to core infrastructure during the requirements phase. The effects on infrastructure changes, which influence or drive the development process. Core infrastructure changes impact the development cycle in many ways and lead derived requirements. If derived requirements based on commercial market influences the capability cycle, program managers could account for the cost and schedule impacts early on in a program's life cycle. Today these changes to the infrastructure are difficult to capture as core requirements. The question is of how to add infrastructure capabilities into the requirements process remains challenging.

A second question is how infrastructure influences the engineering life cycle. For example, research is required on how to move the derived requirements of infrastructure (e.g., security) to the left of the engineering "V" concept phase. What is the impact of adding commercial standards and commercial-like, infrastructure-derived requirements into the acquisition life cycle before the engineers get the user requirements?

A third question is what related to software ecosystems, could DOD align software ecosystems follow the same software development standards. This research would help to define the different strategies incorporated by each service today and what would need to change if they all only had one ecosystem to follow. Would this result in cost savings or avoidance, or would the change be so disruptive that it would be detrimental to the acquisition processes in place?

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Android. 2014. "Android Developers." Accessed May 21, 2014. http://developer.android.com.

Apple. 2013. "Start Developing iOS Apps Today." Last modified December 19. https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/index.html#//apple_ref/doc/uid/TP40011343.

Arimura, Yuji, and Masako Ito. 2011. "Cloud Computing for Software Development Environment." *Fujitsu Science Technical Journal* 47(3): 325–324. http://www.fujitsu.com/downloads/MAG/vol47-3/paper12.pdf.

Bias, Randy. 2008. "Clouds Are Not Datacenters," *Cloud Scaling*. October 8. http://www.cloudscaling.com/blog/cloud-computing/clouds-are-not-datacenters/.

Blackwell, Keith. 2005. "Software Developer's Kit (SDK)," *WhatIs.com.* Accessed January 2014. http://whatis.techtarget.com/definition/software-developers-kit-SDK/.

Bosch. 2013. "Achieving Simplicity with the Three-Layer Product Model." *Computer* 46(11):34–39. doi 10.1109/MC.2013.295.

Bosch. 2009. "From Software Product Lines to Software Ecosystem." Paper presented at the 13th International Software Product Line Conference, San Francisco, California, August 24–26. http://www.janbosch.com/jan_bosch/Composition_files/SPLC09-SoftwareEcosystems-Accepted.pdf.

Brownsword, Lisa, Cecila Albert, David J Carney, Patrick R. Place, Charles Hammons, and John J.Hudak. 2013. "Isolating Patterns of Failure in Department of Defense Acquisition," *Software Engineering Institute Technical Note CMU//SEI 2013-TN-014*. Accessed 23 August 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=53252.

Cyber Security and Information Systems Information Analysis Center. 2013. DOD Cybersecurity Policy Chart, accessed 15 May 2014 http://iac.dtic.mil/csiac/ia_policychart.html.

Defense Acquisition System. 2015. DOD Instruction 5000.02, accessed 5 Feb 2015 https://dap.dau.mil/aphome/das/Pages/Default.aspx.

Dittmer, John. 2010. "Charting a Course for Information Assurance Policy," *CHIPS-Department of the Navy's Information Technology Magazine Online* (Oct-Dec 2010): accessed 24 June 2014, http://www.doncio.navy.mil/chips/ArticleDetails.aspx?ID=2342.

Garlan, David and Mary Shaw, 1994. "An Introduction to Software Architecture," *School of Computer Science. Carnegie Mellon University, Technical Report No. CMU/SEI-94TR-21*, accessed 25January 2014 https://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf.

Henneke, Cameron. 2013. "Android vs. iOS: Comparing the Development Process of the GQueues Mobile Apps," *GQueues Blog*, accessed 20 June 2014 http://blog.gqueues.com/2013/07/android-vs-ios-comparing-development.html.

Interoperable Open Architecture. 2012. "Architecting for Interoperability," *Real-Time Innovations (RTI)*, accessed 1 July 2014 http://www.rti.com/whitepapers/Interoperable_Open_Architecture.pdf.

Jansen, Slinger and Michael Cusumano. 2012. "Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance," *Proceedings of IWSECO 2012*, 879(4): 41–58, accessed 2 March 2014 http://ceur-ws.org/Vol-879/paper4.pdf.

Janssen, Cory. 2014a. "JavaBeans," *Techopedia*, accessed 14 May 2014 http://www.techopedia.com/definition/7865/javabeans.

Janssen, Cory. 2014b. "Portability," *Techopedia*, accessed 7 May 2014 http://www.techopedia.com/definition/8921/portability.

Jayaraman, Karthik. 2013. "Factors Influencing the Success of Platform Centric Ecosystem Strategies: A Case Study of Google Android," *The International Conference on E-Technologies and Business on the Web*, 212–217, accessed 16 Feb 2014 http://sdiwc.net/digital-library/factors-influencing-the-success-of-platform-centric-ecosystem-strategies-a-case-study-of-google-android.

Joachim, Nils, and Daniel Beimborn, Tim Weitzel. 2012. "The Influence of SOA Governance Mechanisms on IT Flexibility and Service Reuse," *The Journal of Strategic Information Systems*, 22(1): 86–101. doi:10.1016/j.jsis.2012.10.003.

Joint Capabilities Integration and Development System Policy (JCIDS). 2012. *Defense Acquisition Guidebook*, accessed 13 June 2014 https://dap.dau.mil/aphome/jcids/Pages/Default.aspx.

Joint Interoperability Test Center (JITC). 2014. "Joint Interoperability Test Center (JITC) Interoperability Certification testing," Last modified June 2014. https://dap.dau.mil/acquipedia/Pages/ArticleDetails.aspx?aid=fb7a9b64-83cd-4c1d-9147-c32b4ae9a556.

Joint Requirements Oversight Council (JROC). 2014. *Defense Acquisition Guidebook*, accessed 13 June 2014 https://acc.dau.mil/CommunityBrowser.aspx?id=518696#10.2.2.

Kundra, Vivek. 2010. "25 Point Implementation Plan to Reform Federal Information Technology Management," *U.S. Chief Information Office*, accessed 15 May 2014 https://www.dhs.gov/sites/default/files/publications/digital-strategy/25-point-implementation-plan-to-reform-federal-it.pdf.

Lam, Wing. 2007. "Information Systems Integration and Enterprise Application Integration (EAI) Adoption: a Case from Financial Services," *Journal of Information Systems Education*, 18(2): 149–157, accessed 30 Sept. 2013 http://eric.ed.gov/?id=EJ832844.

Leiner, Barry, Vinton Cerf, David Clark, Robert Kahn, Leonard Kleinrock, David Lynch, Jon Postel, Larry Roberts, and Stephen Wolff. 2012. "Brief History of the Internet," *Internet Society Brochure*, accessed 1 Oct 2014 http://www.internetsociety.org/brief-history-Internet.

Longworth, David. 2005. "Weak Governance Haunts SOAs," *Loosely Coupled*, 6: 1, 10–12, accessed 25 Jan. 2014 http://www.looselycoupled.com/pubs/digest/LCdigest_200505_p1.pdf.

Lyle, Amaani. 2013. "Pentagon Official Outlines Advantages of Open Architecture," *American Forces Press Service*, accessed 20 June 2014. http://www.defense.gov/news/newsarticle.aspx?id=121103.

Marko, Kurt. 2013. "Why You Need an App Store," *InformationWeek*, 1369: 26–29, accessed 30 Sept. 2013. http://search.proquest.com/docview/1398488758?accountid=12702.

McDonald, Christopher. 2010. "From Art Form to Engineering Discipline? A History of U.S. Military Software Development Standards, 1974–1998," *IEEE Annals of the History of Computing*, 32(04): 32–47, accessed 12 Feb 2014 http://doi.ieeecomputersociety.org/10.1109/MAHC.2009.58.

Mooney, James D. 2004. "Developing Portable Software," *Department of Computing Science and Electrical Engineering, West Virginia University, CS533 Developing Portable Software course*, accessed 30 June 2014 http://www.cs.colostate.edu/saxs/researchexam/DevelopingPortableSoftware.pdf.

Network World. 2013. "A Strategic Approach to Cloud Integration." Accessed November 26, 2014. http://search.proquest.com/docview/1417566740?accountid=12702.

Naur, Peter, and Brian Randell. 1969. "Software Engineering," *Report on a conference by the NATO Science Committee, Garmisch, Germany, 7th to 11th Oct 1968*, accessed 6 Nov. 2013 http://www.scrummanager.net/files/nato1968e.pdf.

Office of Secretary of Defense. 2010. "A New Approach for Delivering Information Technology Capabilities in the Department of Defense," *Report to Congress*, accessed 21 May 2014 http://dcmo.defense.gov/documents/OSD%2013744-10%20-%20804%20Report%20to%20Congress%20.pdf.

Open Systems Architecture (OSA). 2014. Accessed 20 May 2014 http://www.acq.osd.mil/se/initiatives/init_osa.html.

Oracle, 2010. "The JAVA EE 5 Tutorial," *Oracle Documentation*, accessed 20 May 2014 https://docs.oracle.com/javaee/5/tutorial/doc/bnabo.html#bnabq.

Planning, Programming Budgeting and Execution (PPBE) Process. 2012. Accessed 21 June 2014 https://dap.dau.mil/aphome/ppbe/Pages/Default.aspx.

Serbu, Jared. 2013. "DOD Brings Culture of Open Architecture to a World of Proprietary Systems," *Federal News Radio*, accessed 20 June 2014 http://www.federalnewsradio.com/394/3504114/DOD-brings-culture-of-open-architecture-to-a-world-of-proprietary-systems.

Serbu, Jared. 2014. "Navy Issues First Contract under New Datacenter Consolidation Plan," *Federal News Radio*, accessed 28 June 2014 http://www.federalnewsradio.com/412/3640559/Navy-issues-first-contract-under-new-data-center-consolidation-plan.

Shea, John. 2009. "DOD Transformation to Service-oriented Information Enterprise via Net-centric Strategies," *Brief at AV SOA Conference, 2009*, accessed 16 Nov 2014 http://www.acq.osd.mil/damir/2009%20Conference/Service%20Oriented%20Enterprise%20Shea.pdf.

Shebalin, Paul. 1994. "Software Development Standards and the DOD Program Manager," *Acquisition Review Quarterly, summer 1994*, accessed 11 Jul 2014 http://www.dau.mil/pubscats/pubscats/AR%20Journal/arq94/shebali.pdf.

Takai, Teresa. 2013. "Joint Information Environment Implementation Guidance," *Department of Defense Memorandum*, accessed 25 Jan 2014 http://dodcio.defense.gov/Portals/0/Documents/JIE/20130926_Joint%20Information%20Environment%20Implementation%20Guidance_DOD%20CIO_Final_Document.pdf .

TechTerms. 2013. Framework. Last modified March 7 2013. http://www.techterms.com/definition/framework.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California